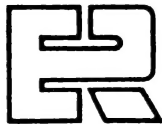


B.GEOFFRION-R.WEISS

50
PROGRAMMES ASSEMBLEUR
TO 7-70

S. E. C. F.



Editions Radio

9, RUE JACOB - 75006 PARIS
TEL. 329.63.70

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1^{er} de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal.

© SECF Éditions Radio, Paris 1985 <i>Tous droits de traduction, de reproduction et d'adaptation réservés pour tous pays.</i>	Imprimé en France par Berger-Levrault, Nancy
	Dépôt légal : avril 1985 Éditeur n° 998 - Imprimeur : 778710 I.S.B.N. 2 7091 0966 2

INTRODUCTION

Les lecteurs, utilisateurs ou non du matériel Thomson (TO7 ou TO7-70) trouveront ici un ensemble de programmes didactiques ou (et) ludiques leur permettant de s'initier au langage assembleur du 6809 — langage le mieux adapté pour l'écriture de programmes courts et rapides puisque le plus proche de la machine.

Les listages donnés sont obtenus à partir du TO7 et les programmes font parfois appel aux « routines » du moniteur dont le tableau I fournit la liste avec leur « nom ».

Les problèmes traités sont de difficulté croissante et les programmes montrent l'utilisation de telle ou telle instruction sans chercher « l'astuce » qui ferait « gagner » 1 ou 2 octets.

Nous conseillons au lecteur désirant écrire ses programmes de commencer par des problèmes **très** simples qui une fois résolus lui permettront d'aborder des problèmes un peu plus complexes. Surtout ne vous aventurez pas dans l'écriture de programmes exigeant plusieurs centaines d'octets — tentation existant en particulier chez les habitués du BASIC qui croient avoir appris à programmer : **l'assembleur n'est pas un langage évolué !**

Si vous écrivez vous-même les programmes proposés, peut-être aurez-vous un autre listage que le nôtre. Si votre programme — après quelques mises au point — « marche », vous avez gagné. Comparez les solutions, comprenez les différences (en bien ou en mal) et passez à la suite. Ce qu'il faut **surtout** faire, dès lors que l'on utilise l'assembleur, c'est :

- écrire des programmes **courts**
- les mettre au point grâce :
 - au pas à pas
 - au point d'arrêt
 - à l'examen des cases mémoires
 - au test pour **diverses** valeurs des variables
- réunir ces programmes soit en écriture « linéaire » soit à l'aide de « sous-programmes » pour constituer un logiciel conséquent
- **commenter** abondamment le programme source.

Nous rappelons dans le tableau II les instructions du 6809 avec les divers modes d'adressage, et les flags qu'elles affectent. Pour plus de détails, consultez les ouvrages des constructeurs.

Les premiers programmes proposés ont été écrits pour que vous vous familiarisiez avec le code hexadécimal.

Suit une série de programmes spécialement conçus pour expliciter le fonctionnement du 6809 et l'usage de telle ou telle instruction.

Puis nous vous proposons des programmes de plus en plus longs qui sont essentiellement des jeux.

Enfin, nous vous suggérons quelques idées, en vous donnant le fil conducteur. Si vous désirez avoir une solution, utilisez le bon que vous trouverez à la fin de l'ouvrage.

Tableau I : Les sous-programmes moniteur et leurs adresses

Nom	Adresse	Utilisation
PUTC	E803	Affichage d'un caractère (B)
GETC	E806	Lecture clavier
KTST	E809	Lecture rapide du clavier
DRAW	E80C	Tracé d'un segment de droite
PLOT	E80F	Allumage ou extinction d'un point
RCOS	E812	Gestion interface de communication
K7CO	E815	Lecture/écriture sur cassette
GETL	E818	Lecture du photostyle
LPIN	E818	Lecture du contact du photostyle
NOTE	E81E	Génération d'une note
GETP	E821	lecture de la couleur d'un point
GETS	E824	Lecture d'un caractère à l'écran
JOYS	E827	Lecture manettes de jeu
DKCO	E82A	Contrôleur de disquettes
MENU	E82D	Retour au menu
KBIN	E830	Sortie programme d'interruption
CHPL	E833	Ecriture d'un point « caractère »

Tableau II : Instructions du 6809

[illegible]

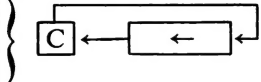
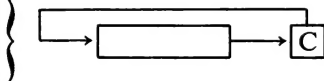
Tableau II : Instructions du 6809 (suite)

Mnémo- nique	Opération réalisée	Flags					Modes d'adressage					
		H	N	Z	V	C	lh	lm	Di	Et	In	Re
BCS LBCS	Branchement si C = 1											* *
BEQ LBEQ	Branchement si Z = 1 (ou égal, ou nul)											* *
BGE LBGE	Branchement si \geq (nombres signés)											* *
BGT LBGT	Branchement si $>$ (nombres signés)											* *
BHI LBHI	Branchement si $>$ (nb. non signés)											* *
BHS LBHS	Branchement si \geq (nb. non signés)											* *
BITA BITB	Test log. (A)ET(M) Test log. (B)ET(M)		*	*	\emptyset			*	*	*	*	
			*	*	\emptyset			*	*	*	*	
BLE LBLE	Branchement si \leq (nombres signés)											* *
BLO LBLO	Branchement si $<$ (nb. non signés)											* *
BLS LBLS	Branchement si \leq (nb. non signés)											* *
BLT LBLT	Branchement si $<$ (nombres signés)											* *
BMI LBMI	Branchement si N = 1 (négatif, <0)											* *
BNE LBNE	Branchement si Z = \emptyset (différent de)											* *
BPL LBPL	Branchement si N = \emptyset (positif, >0)											* *
BRA LBRA	Branchement Inconditionne											* *
BSR LBSR	Branchement à un sous-programme											* *
BVC LBVC	Branchement si V = \emptyset											* *
BVS LBVS	Branchement si V = 1											* *
CLRA CLRB CLR	$\emptyset \rightarrow (A)$ $\emptyset \rightarrow (B)$ $\emptyset \rightarrow (M)$		\emptyset \emptyset \emptyset	1 1 1	\emptyset \emptyset \emptyset	\emptyset \emptyset \emptyset	* * *					
									*	*	*	

Tableau II : Instructions du 6809 (suite)

Mnémo- nique	Opération réalisée	Flags					Modes d'adressage						
		H	N	Z	V	C	Ih	Im	Di	Et	In	Re	
CMPA CMPB CMPD CMPS CMPU CMPX CMPY	Compare (A) à (M) Compare (B) à (M) Comp.(D) à (M:M + 1) Comp.(S) à (M:M + 1) Comp.(U) à (M:M + 1) Comp.(X) à (M:M + 1) Comp.(Y) à (M:M + 1)	? ?	*	*	*	*		*	*	*	*		
COMA COMB COM	Complément à 1 (A) Complément à 1 (B) Complément à 1 (M)		*	*	0	1	*						
CWAI	(CC)ETdata→(CC) attente interruption	*	*	*	*	*	*						
DAA	Ajustement dec.(A)		*	*	0	*	*						
DECA DECB DEC	(A) – 1→(A) (B) – 1→(B) (M) – 1→(M)		*	*	*		*						
EORA EORB	(A) + (M)→(A) (B) + M→(B)		*	*	0		*						
EXG R ₁ ,R ₂	(R ₁)↔(R ₂)						*						
INCA INCB INC	(A) + 1→(A) (B) + 1→(B) (M) + 1→(M)		*	*	*		*						
JMP	A.E.→(PC)								*	*	*		
JSR	Saut à sous-programme								*	*	*		
LDA LDB LDD LDS LDU LDX LDY LEAS LEAU LEAX LEAY	(M)→(A) (M)→(B) (M:M + 1)→(D) (M:M + 1)→(S) (M:M + 1)→(U) (M:M + 1)→(X) (M:M + 1)→(Y) A.E.→(S) A.E.→(U) A.E.→(X) A.E.→(Y)		*	*	0			*	*	*	*		
LSLA LSLB LSL	<div><div><div>C</div><div>←</div><div>←</div><div>←</div><div>0</div></div><div>mul. par 2 signée</div></div>		*	*	*	*	*	*					
LSRA LSRB LSR	<div><div><div>0</div><div>→</div><div>→</div><div>→</div><div>C</div></div><div>div. par 2 non signée</div></div>		0	*		*	*	*					
MUL	(A) *(B)→(D) non signée			*		*	*	*					
NEGA NEGB NEG	Complément à 2 (A) Complément à 2 (B) Complément à 2 (M)	? ? ?	*	*	*	*	*						

Tableau II : Instructions du 6809 (suite)

Mnémo- nique	Opération réalisée	Flags						Modes d'adressage				
		H	N	Z	V	C	Ih	Im	Di	Et	In	Re
NOP	Pas d'opération						*					
ORA	(A)OU(M)→(A)		*	*	0		*					
ORB	(B)OU(M)→(B)		*	*	0		*					
ORCC	(CC)OUdata→(CC)	*	*	*	*	*	*					
PSHS	Sauve. en pile S						*					
PSHU	Sauve. en pile U						*					
PULS	Rest. de pile S						*					
PULU	Rest. de pile U						*					
ROLA			*	*	*	*	*					
ROLB			*	*	*	*	*					
ROL			*	*	*	*	*		*	*	*	
RORA			*	*	*	*	*					
RORB			*	*	*	*	*					
ROB			*	*	*	*	*		*	*	*	
RTI	Retour interrupt.	*	*	*	*	*	*					
RTS	Retour sous-prog						*					
SBCA	(A)-(M)-C→(A)	?	*	*	*	*		*	*	*	*	
SBCB	(B)-(M)-C→(B)	?	*	*	*	*		*	*	*	*	
SEX	Ext. sig. (B)→(D)		*	*	0		*					
STA	(A)→(M)		*	*	0			*	*	*	*	
STB	(B)→(M)		*	*	0			*	*	*	*	
STD	(D)→(M:M+1)		*	*	0				*	*	*	
STS	(S)→(M:M+1)		*	*	0				*	*	*	
STU	(U)→(M:M+1)		*	*	0				*	*	*	
STX	(X)→(M:M+1)		*	*	0				*	*	*	
STY	(Y)→(M:M+1)		*	*	0				*	*	*	
SUBA	(A)-(M)→(A)	?	*	*	*	*			*	*	*	
SUBB	(B)-(M)→(B)	?	*	*	*	*			*	*	*	
SUBD	(D)-(M:M+1)→(D)		*	*	*	*		*	*	*	*	
SWI	Inter. logicielle						*					
SWI2							*					
SWI3							*					
SYNC	Synchro. interruption						*					
TFR R ₁ ,R ₂	(R ₁)→(R ₂)							*				
TSTA	Test		*	*	0		*					
TSTB			*	*	0		*					
TST			*	*	0				*	*	*	

NOTES

— Adressages

Ih: Inhérent, Im: Immédiat, Di: Direct,
Et: Etendu, In: Indéxé, Re: Relatif.— R₁,R₂

soit A,B,DP,CC

soit D,S,U,X,Y,PC

— ? flag affecté mais non significatif

— A.E. Adresse Effective

— (M:M+1) : Mot de 16 bits stocké en M
(poids fort) et M+1.

TABLEAU III
Codes ASCII - Codes hexadécimaux et correspondance

00 NUL	10 DLE	20 SP	30 0	40 @	50 P	60 '	70 p
01 SOH	11 DLE	21 !	31 1	41 A	51 Q	61 a	71 q
02 STX	12 DC2	22 "	32 2	42 B	52 R	62 b	72 r
03 ETX	13 DC3	23 #	33 3	43 C	53 S	63 c	73 s
04 EOT	14 DC4	24 \$	34 4	44 D	54 T	64 d	74 t
05 ENO	15 NAK	25 %	35 5	45 E	55 U	65 e	75 u
06 ACK	16 SYN	26 &	36 6	46 F	56 V	66 f	76 v
07 BEL	17 ETB	27 '	37 7	47 G	57 W	67 g	77 w
08 BS	18 CAN	28 (/	38 8	48 H	58 X	68 h	78 x
09 HT	19 EM	29)	39 9	49 I	59 Y	69 i	79 y
0A LF	1A SUB	2A *	3A :	4A J	5A Z	6A j	7A z
0B VT	1B ESC	2B +	3B ;	4B K	5B [6B k	7B {
0C FF	1C FS	2C ,	3C <	4C L	5C \	6C l	7C
0D CR	1D GS	2D -	3D =	4D M	5D]	6D m	7D }
0E SO	1E RS	2E .	3E >	4E N	5E ^	6E n	7E ~
0F SI	1F US	2F /	3F ?	4F O	5F _	6F o	7F DEL

Les codes de 00 à 1F sont des codes de contrôle, leur signification dépend parfois du constructeur, nous utiliserons :

- 07 BEL ou BIP sonore
- 08 BS déplacement arrière du curseur (*Back Space*)
- 09 HT Tabulation ou déplacement avant
- 0A LF Saut de ligne (*Line Feed*)
- 0B VT Déplacement vers le haut
- 0D CR Retour Chariot (*Carriage Return*)
- 1B ESC Escape
- 1F US Unit Separator

Ces 2 derniers codes seront utilisés pour le contrôle de l'écran.

TROIS OPÉRATIONS EN HEXADÉCIMAL

BUT : Vous familiariser avec l'hexadécimal et les commandes de votre ordinateur.

La meilleure façon de se familiariser avec un code que l'on ne maîtrise pas complètement, c'est bien entendu de le pratiquer.

Mais comment le pratiquer si on ne connaît pas encore la machine qui nous permettra de l'utiliser ?

C'est pourquoi, pour commencer, nous vous proposons une série de programmes, relativement compliqués en tant que réalisation mais qu'il n'est pas nécessaire d'analyser et de comprendre pour les exploiter.

Quand vous serez devenu un véritable spécialiste de l'assembleur 6809 (c'est-à-dire à la fin du livre !), vous pourrez toujours revenir sur leur réalisation pour en analyser le fonctionnement.

Le premier programme est un programme d'addition hexadécimale sur 1 digit. Le système vous demande de rentrer des opérateurs hexadécimaux et rejette les caractères non autorisés par exemple K,L,V etc... puis vous affiche le résultat sur l'écran.

Le second est également un programme d'addition hexadécimale mais cette fois sur 4 digits. Son fonctionnement est le même. Toutefois, remarquez la progression de la retenue et le risque d'erreur quand le résultat nécessitera 5 digits.

Le troisième est un programme de soustraction sur 4 digits.

Notez : Un digit est un mot binaire de 4 bits exprimé en fonction de sa valeur par un des caractères suivants : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Le digit de rang n vaut 16 fois plus que le digit de rang n-1.

Exemple : 10 hex égal à 16 en décimal

42 hex égal à $(16 * 4) + 2$ soit 68 en décimal

Exercez-vous : Combien valent en décimal

38 hex, A8 hex, 100 hex, A654 hex ?

1

ADDITION HEXADÉCIMALE SUR 1 DIGIT

```

                                * ADDITION HEXADÉCIMALE (1digit)

                                E833    CHPL    EQU    $E833
                                E80C    DRAW    EQU    $E80C
                                6041    CHDRAW  EQU    $6041
                                E806    GETC    EQU    $E806
                                E803    PUTC    EQU    $E803
                                0000    CR      EQU    $00
7800 2A 41 44 44 44    MESST    FCC    /*ADDITION HEXADÉCIMALE*/
7804 49 54 49 4F
7808 4E 20 48 45
780C 58 41 44 45
7810 43 49 4D 41
7814 4C 45 2A
7817 04
7818 20 20 20 20    MESS0    FCB    $04    delimitteur
781C 20 20 20 20    FCC    /
7820 20 20 20 20
7824 20 20 20 20
7828 20 20 20 20
782C 20 20 20 20
7830 20 20 20 20    FCC    /
7834 20 20 20 20
7838 20 20 20 20
783C 20 20 20 20
7840 20
7841 04    FCB    $04
```

7842	46	72	61	70	MESS1	FCC	/Frappez un caractere HEX/
7846	70	65	7A	20			
784A	75	6E	20	63			
784E	61	72	61	63			
7852	74	65	72	65			
7856	20	48	45	58			
785A	44	45	43	49		FCC	/DECIMAL/
785E	40	41	40				
7861	04					FCB	\$04

7862	4E	6F	75	76	MESS2	FCC	/Nouvelle addition ---> /
7866	65	60	60	65			
786A	20	61	64	64			
786E	69	74	69	6F			
7872	6E	20	20	20			
7876	20	20	3E	20			
787A	20	45	4E	54		FCC	/ ENTREE /
787E	52	45	45	20			
7882	04					FCB	\$04
7883	52	65	74	6F	MESS3	FCC	/Retour au moniteur ---> /
7887	75	72	20	61			
788B	75	20	60	6F			
788F	6E	69	74	65			
7893	75	72	20	20			
7897	20	20	3E	20			
789B	20	75	6E	65		FCC	/ une TOUCHE/
789F	20	54	4F	55			
78A3	43	48	45				
78A6	04					FCB	\$04
		0010			NVAL	EQU	16T
78A7	30	31	32	33	VAL	FCC	/0123456789ABCDEF/
78AB	34	35	36	37			
78AF	38	39	41	42			
78B3	43	44	45	46			
78B7	00					FCB	OR
78B8	CE	0000			ADD	LDU	#ENDMEM

78BB	06	62			LDB	##62	tour	vert	← Couleur du tour
78BD	BD	7A2A			JSR	MATT			

78C0	06	6B			LDB	##6B			
78C2	BD	7A2A			JSR	MATT			

78C5	86	24			LDA	##24	LIGNE BASSE	← Positionnement de la fenêtre sur la
78C7	06	00			LDB	##00	LIGNE HAUTE	totalité de l'écran
78C9	BD	79CB			JSR	POFEN		

78CC	06	0C			LDB	##0C		← Effacement de la totalité de l'écran
78CE	BD	E803			JSR	PUTC		

78D1	86	0A		LDA	#10T	colonne	← MESSAGE titre
78D3	06	00		LDB	#00T	ligne	
78D5	8D	7A14		JSR	POUR		
78D8	8E	7800		LDX	#MESS1		
78DB	8D	7A37		JSR	MESSAG		
78DE	86	24		LDA	#\$24	LIGNE BASSE	← Positionnement de la fenêtre sur la
78E0	06	06		LDB	#\$06	LIGNE HAUTE	partie devant être modifiée
78E2	8D	790B		JSR	POFEN		
78E5	06	0C	NADD	LDB	#\$0C		← Effacement de la fenêtre
78E7	8D	E803		JSR	PUTC		
78EA	86	01		LDA	#01		← MESSAGE 1
78EC	06	18		LDB	#24T		
78EE	8D	7A14		JSR	POUR		
78F1	8E	7842		LDX	#MESS1		
78F4	8D	7A37		JSR	MESSAG		
78F7	86	14		LDA	#20T	COLONNE	← PREMIER OPERANDE
78F9	06	0A		LDB	#10T	LIGNE	
78FB	8D	7A06		JSR	POUDC		
78FE	8D	7A95	CONT	JSR	CAR		
7901	8D	7A7C		JSR	COMPAR		
7904	26	02		BNE	CTOUR	COULEUR TOUR	
7906	20	10		BRA	AFF		
7908	06	61	CTOUR	LDB	#\$61	TOUR ROUGE	
790A	8D	7A2A		JSR	MATT		
790D	8D	79BF		JSR	PAUSE		
7910	06	62		LDB	#\$62	TOUR VERT	
7912	8D	7A2A		JSR	MATT		
7915	7E	78FE		JMP	CONT		
7918	8D	E803	AFF	JSR	PUTC	AFFICHAGE	
791B	8D	7A73		JSR	ASCHEX		
791E	36	04		PSHU	B		
7920	86	10		LDA	#16T	COLONNE	← SIGNE
7922	06	0E		LDB	#14T	LIGNE	
7924	8D	7A06		JSR	POUDC		
7927	06	2B		LDB	#'+		
7929	8D	E803		JSR	PUTC		
792C	86	14		LDA	#20T	COLONNE	← SECOND OPERANDE
792E	06	0E		LDB	#14T	LIGNE	
7930	8D	7A06		JSR	POUDC		
7933	8D	7A95	CONT1	JSR	CAR		
7936	8D	7A7C		JSR	COMPAR		
7939	26	02		BNE	CTOUR1		
793B	20	10		BRA	AFF1		

793D	06	61	CTOUR1	LDB	##61	TOUR ROUGE	
793F	BD	7A2A		JSR	MATT		
7942	BD	798F		JSR	PAUSE		
7945	06	62		LDB	##62	TOUR VERT	
7947	BD	7A2A		JSR	MATT		
794A	7E	7933		JMP	CONT1		
794D	BD	E803	AFF1	JSR	PUTC	AFFICHAGE	
7950	BD	7A73		JSR	ASCHEX		
7953	EB	04		ADDB	.U		
7955	36	04		PSHU	B		
7957	8E	0010		LDX	#16T	←	TRAIT
795A	108E	0010		LDY	#16T		
795E	06	2D		LDB	#'-		
7960	F7	6041		STB	CHDRAW		
7963	BD	E833		JSR	CHPL		
7966	8E	0018		LDX	#24T		
7969	108E	0010		LDY	#16T		
796D	BD	E80C		JSR	DRAW		
7970	86	12		LDA	#18T	←	RESULTAT
7972	06	12		LDB	#18T	COLONNE	LIGNE
7974	BD	7A06		JSR	PCUDC		
7977	37	04		PULU	B		
7979	BD	7A50		JSR	HEXASC		
797C	BD	E803		JSR	PUTC		
797F	1F	89		TFR	A,B		
7981	BD	E803		JSR	PUTC		
7984	BD	7A43		JSR	TANOR	TAILLE NORMALE	
7987	33	41		LEAU	.U		
7989	86	01		LDA	#01	←	MESSAGE 0
798B	06	18		LDB	#24T	colonne	ligne
798D	BD	7A14		JSR	PCUR		
7990	8E	7818		LDX	#MESS0		
7993	BD	7A37		JSR	MESSAG		
7996	86	01		LDA	#01	←	MESSAGE 2
7998	06	17		LDB	#23T		
799A	BD	7A14		JSR	PCUR		
799D	8E	7862		LDX	#MESS2		
79A0	BD	7A37		JSR	MESSAG		
79A3	86	01		LDA	#01	←	MESSAGE 3
79A5	06	18		LDB	#24T		
79A7	BD	7A14		JSR	PCUR		
79AA	8E	7883		LDX	#MESS3		
79AD	BD	7A37		JSR	MESSAG		
79B0	BD	7A95		JSR	CAR		
79B3	01	0D		CMPB	#CR		
79B5	1027	FF2C		LBEQ	NADD		
79B9	06	1E		LDB	##1E	en haut gauche	
79BB	BD	E803		JSR	PUTC		
79BE	3F			SWI			

*****SOUS-PROGRAMMES*****

79BF	36	10	PAUSE	PSHU	X
79C1	8E	FFFF		LDX	#\$FFFF
79C4	30	1F	ENC	LEAX	-1,X
79C6	26	FC		BNE	ENC
79C8	37	10		PULU	X
79CA	39			RTS	

PAUSE

79CB	36	06	POFEN	PSHU	A,B
79CD	06	1F		LDB	#\$1F
79CF	BD	E803		JSR	PUTC
79D2	E6	C4		LDB	.U
79D4	54			LSRB	
79D5	54			LSRB	
79D6	54			LSRB	
79D7	54			LSRB	
79D8	C4	1F		ANDB	#\$1F
79DA	CA	10		ORB	#\$10
79DC	BD	E803		JSR	PUTC
79DF	E6	C4		LDB	.U
79E1	C4	1F		ANDB	#\$1F
79E3	CA	10		ORB	#\$10
79E5	BD	E803		JSR	PUTC
79E8	C6	1F		LDB	#\$1F
79EA	BD	E803		JSR	PUTC
79ED	E6	41		LDB	1.U
79EF	54			LSRB	
79F0	54			LSRB	
79F1	54			LSRB	
79F2	54			LSRB	
79F3	C4	2F		ANDB	#\$2F
79F5	CA	20		ORB	#\$20
79F7	BD	E803		JSR	PUTC
79FA	E6	41		LDB	1.U
79FC	C4	2F		ANDB	#\$2F
79FE	CA	20		ORB	#\$20
7A00	BD	E803		JSR	PUTC
7A03	37	06		PULU	A,B
7A05	39			RTS	

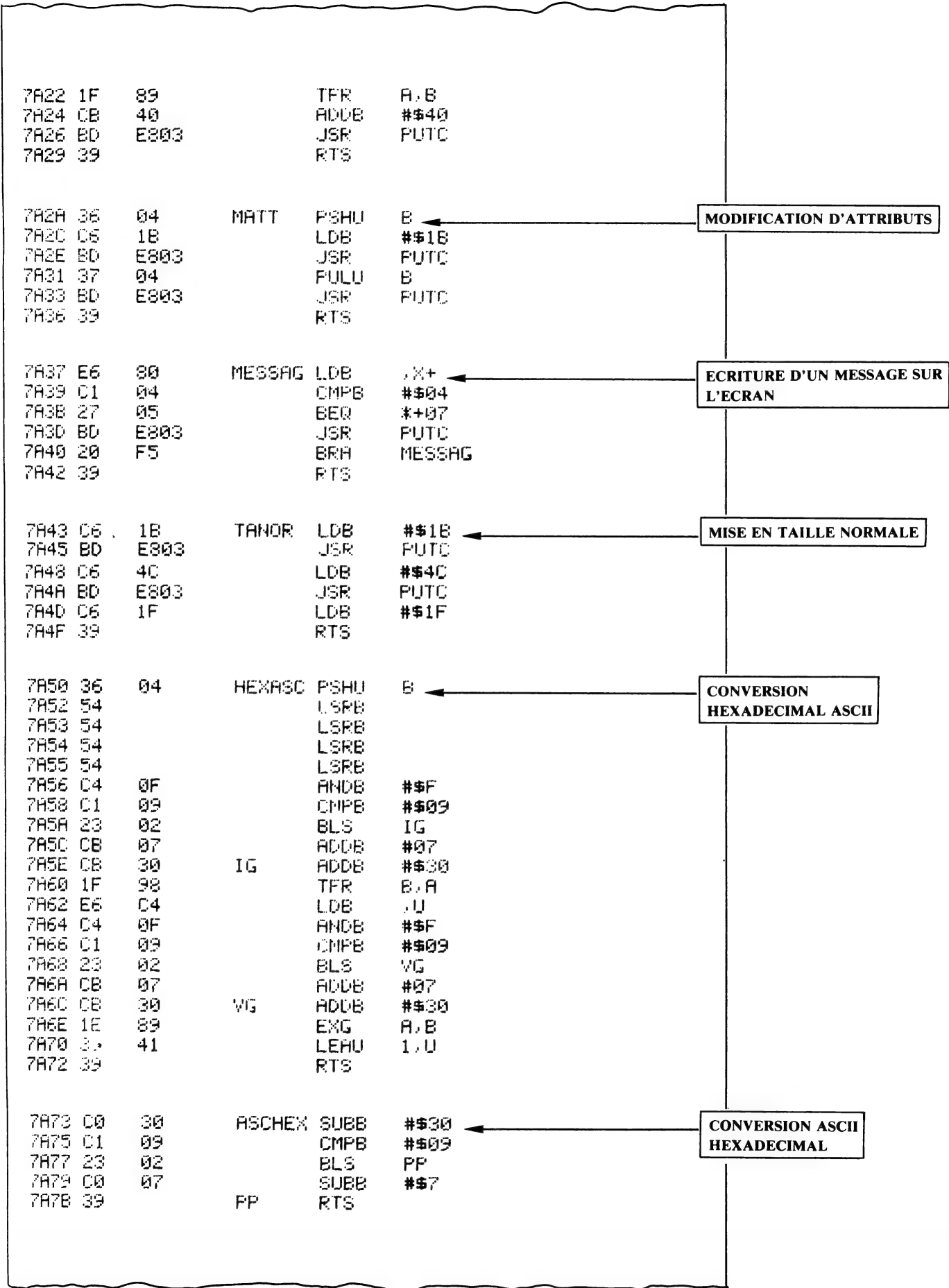
POSITIONNEMENT DE LA FENÊTRE COURANTE

7A06	36	06	POUDC	PSHU	A,B
7A08	C6	18		LDB	#\$18
7A0A	BD	E803		JSR	PUTC
7A0D	C6	4F		LDB	#\$4F
7A0F	BD	E803		JSR	PUTC
7A12	37	06		PULU	A,B

POSITIONNEMENT DU CURSEUR
+ double caractère

7A14	36	06	PCUR	PSHU	A,B
7A16	C6	1F		LDB	#\$1F
7A18	BD	E803		JSR	PUTC
7A1B	37	06		PULU	A,B
7A1D	CB	40		ADDB	#\$40
7A1F	BD	E803		JSR	PUTC

POSITIONNEMENT DU CURSEUR



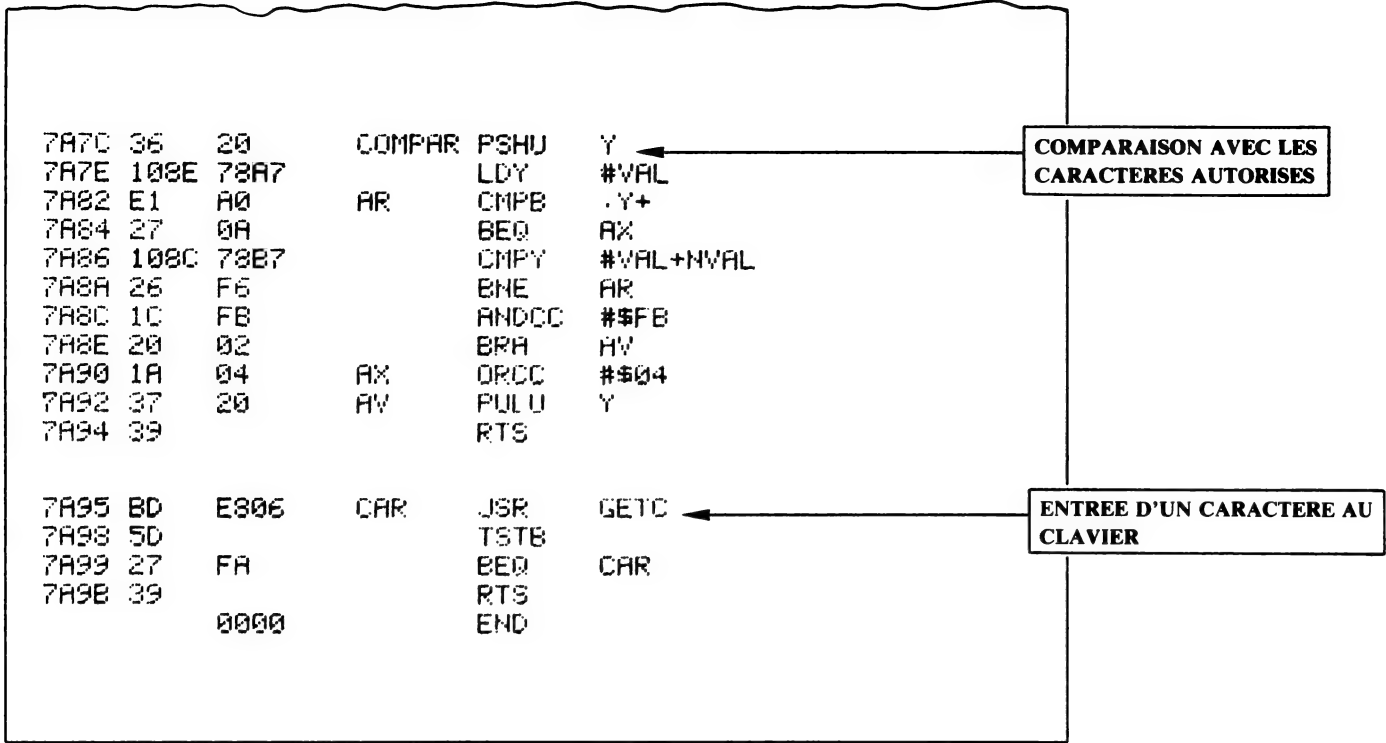


Table des symboles

ADD	3588	HEXASC	3738
AFF	3616	IG	3746
AFF1	3649	MATT	3717
AR	376A	MESS0	3518
ASCHEX	375B	MESS1	3542
AV	377A	MESS2	3562
AX	3778	MESS3	3583
CAR	377D	MESSAG	3722
CHDRAW	2036	MESST	3500
CHPL	0012	NADD	35E4
COMPAR	3764	NVAL	0010
CONT	35FC	PAUSE	36B7
CONT1	362F	PCUDC	36F8
CR	0000	PCUR	3704
CTOUR	3606	POFEN	36C3
CTOUR1	3639	PP	3763
DRAW	000E	PUTC	0002
ENC	36BC	TANOR	372D
GETC	000A	VAL	35A7
		VG	3754

Dans la suite de l'ouvrage, les tables de symboles ne seront plus données.

2

ADDITION HEXADÉCIMALE SUR 4 DIGITS

```

*****
* BG *** ADDITION HEXADÉCIMALE ***RW *
***** sur 4 digits *****

      E809      KTST      EQU      $E809
      E833      CHPL      EQU      $E833
      E80C      DRAW      EQU      $E80C
      6041      CHDRAW    EQU      $6041
      E806      GETC      EQU      $E806
      E803      PUTC      EQU      $E803
      0000      CR        EQU      $00
7B00 2A 41 44 44 MESST FCC      /*ADDITION HEXADÉCIMALE*/
7B04 49 54 49 4F
7B08 4E 20 48 45
7B0C 58 41 44 45
7B10 43 49 4D 41
7B14 4C 45 2A
7B17 04
7B18 20 20 20 20 MESS0 FCB      $04      delimitteur
7B1C 20 20 20 20 FCC      /
7B20 20 20 20 20
7B24 20 20 20 20
7B28 20 20 20 20
7B2C 20 20 20 20
7B30 20 20 20 20 FCC      /
7B34 20 20 20 20
7B38 20 20 20 20
7B3C 20 20 20 20
7B40 20
7B41 04
7B42 46 72 61 70 MESS1 FCB      $04
7B46 70 65 7A 20 FCC      /*Tapez un caractere HEX*/
7B4A 75 6E 20 63
7B4E 61 72 61 63
7B52 74 65 72 65
7B56 20 48 45 58
7B5A 44 45 43 49 FCC      /*DECIMAL*/
7B5E 4D 41 4C
7B61 04 FCC      $04

```

7B62	4E	6F	75	76	MESS2	FCC	/Nouvelle addition ---> /	
7B66	65	6C	6C	65				
7B6A	20	61	64	64				
7B6E	69	74	69	6F				
7B72	6E	20	20	2D				
7B76	2D	2D	3E	20				
7B7A	20	45	4E	54		FCC	/ ENTREE /	
7B7E	52	45	45	20				
7B82	04					FCB	\$04	
7B83	52	65	74	6F	MESS3	FCC	/Retour au moniteur ---> /	
7B87	75	72	20	61				
7B8B	75	20	6D	6F				
7B8F	6E	69	74	65				
7B93	75	72	20	2D				
7B97	2D	2D	3E	20				
7B9B	20	75	6E	65		FCC	/ une TOUCHE/	
7B9F	20	54	4F	55				
7BA3	43	48	45					
7BA6	04					FCB	\$04	
		0010			NVAL	EQU	15T	
7BA7	30	31	32	33	VAL	FCC	/0123456789ABCDEF/	
7BAB	34	35	36	37				
7BAF	38	39	41	42				
7BB3	43	44	45	46				
7BB7	0D					FCB	CR	
7BB8					NDIG	RMB	1	nombre de digit
7BB9					RTENU	RMB	1	RETENUE
7BBA	0E	0000			ADD	LDU	#ENDMEM	
7BB0	06	62				LDB	##62	tour vert ← Couleur du tour
7BBF	BD	7D88				JSR	MATT	
7BC2	06	6B				LDB	##6B	
7BC4	BD	7D88				JSR	MATT	
7BC7	86	24				LDA	##24	LIGNE BASSE ← Positionnement de la fenêtre sur la
7BC9	06	00				LDB	##00	LIGNE HAUTE ← totalité de l'écran
7BCB	BD	7D29				JSR	POFEN	
7BCE	06	0C				LDB	##0C	← Effacement de la totalité de l'écran
7BD0	BD	E803				JSR	PUTC	
7BD3	86	0A				LDA	#10T	colonne ← MESSAGE titre
7BD5	06	00				LDB	#00T	ligne
7BD7	BD	7D72				JSR	PCUR	
7BDA	8E	7B00				LDX	#MESST	
7BDD	BD	7D95				JSR	MESSAG	
7BE0	86	24				LDA	##24	LIGNE BASSE ← Positionnement de la fenêtre sur la
7BE2	06	06				LDB	##06	LIGNE HAUTE ← partie de l'écran devant être modifiée
7BE4	BD	7D29				JSR	POFEN	
7BE7	06	0C			NADD	LDB	##0C	← Effacement de la fenêtre
7BE9	BD	E803				JSR	PUTC	
7BEC	86	01				LDA	#01	← MESSAGE 1
7BEE	06	18				LDB	#24T	
7BF0	BD	7D72				JSR	PCUR	

7BF3 8E	7B42	LDX	#MESS1	
7BF6 BD	7D95	JSR	MESSAG	
7BF9 86	14	LDA	#20T	COLONNE
7BFB 06	0A	LDB	#10T	LIGNE
7BFD BD	7D64	JSR	POUDC	
7C00 06	04	LDB	#\$04	
7C02 F7	7B88	STB	NDIG	
7C05 BD	7DF3	JSR	CAR	
7C08 BD	7DDA	JSR	COMPAR	
7C0B 26	02	BNE	CTOUR	COULEUR TOUR
7C0D 20	10	BRA	AFF	
7C0F 06	61	LDB	#\$61	TOUR ROUGE
7C11 BD	7D88	JSR	MATT	
7C14 BD	7D1D	JSR	PAUSE	
7C17 06	62	LDB	#\$62	TOUR VERT
7C19 BD	7D88	JSR	MATT	
7C1C 7E	7C05	JMP	CONT	
7C1F BD	E803	JSR	PUTC	AFFICHAGE
7C22 BD	7DD1	JSR	ASCHEX	
7C25 36	04	PSHU	B	
7C27 7A	7B88	DEC	NDIG	
7C2A 26	D9	BNE	CONT	
7C2C 86	10	LDA	#16T	COLONNE
7C2E 06	0E	LDB	#14T	LIGNE
7C30 BD	7D64	JSR	POUDC	
7C33 06	2B	LDB	#'+	
7C35 BD	E803	JSR	PUTC	
7C38 86	14	LDA	#20T	COLONNE
7C3A 06	0E	LDB	#14T	LIGNE
7C3C BD	7D64	JSR	POUDC	
7C3F 06	04	LDB	#\$04	
7C41 F7	7B88	STB	NDIG	
7C44 BD	7DF3	JSR	CAR	
7C47 BD	7DDA	JSR	COMPAR	
7C4A 26	02	BNE	CTOUR1	
7C4C 20	10	BRA	AFF1	
7C4E 06	61	LDB	#\$61	TOUR ROUGE
7C50 BD	7D88	JSR	MATT	
7C53 BD	7D1D	JSR	PAUSE	
7C56 06	62	LDB	#\$62	TOUR VERT
7C58 BD	7D88	JSR	MATT	
7C5B 7E	7C44	JMP	CONT1	
7C5E BD	E803	JSR	PUTC	AFFICHAGE
7C61 BD	7DD1	JSR	ASCHEX	
7C64 36	04	PSHU	B	
7C66 7A	7B88	DEC	NDIG	
7C69 26	D9	BNE	CONT1	
7C6B 8E	0010	LDX	#16T	
7C6E 108E	0010	LDY	#16T	
7C72 06	2D	LDB	#'-	
7C74 F7	6041	STB	CHDRAW	
7C77 BD	E833	JSR	CHPL	
7C7A 8E	001C	LDX	#28T	
7C7D 108E	0010	LDY	#16T	
7C81 BD	E80C	JSR	DRAW	

PREMIER OPERANDE

SIGNE

SECOND OPERANDE

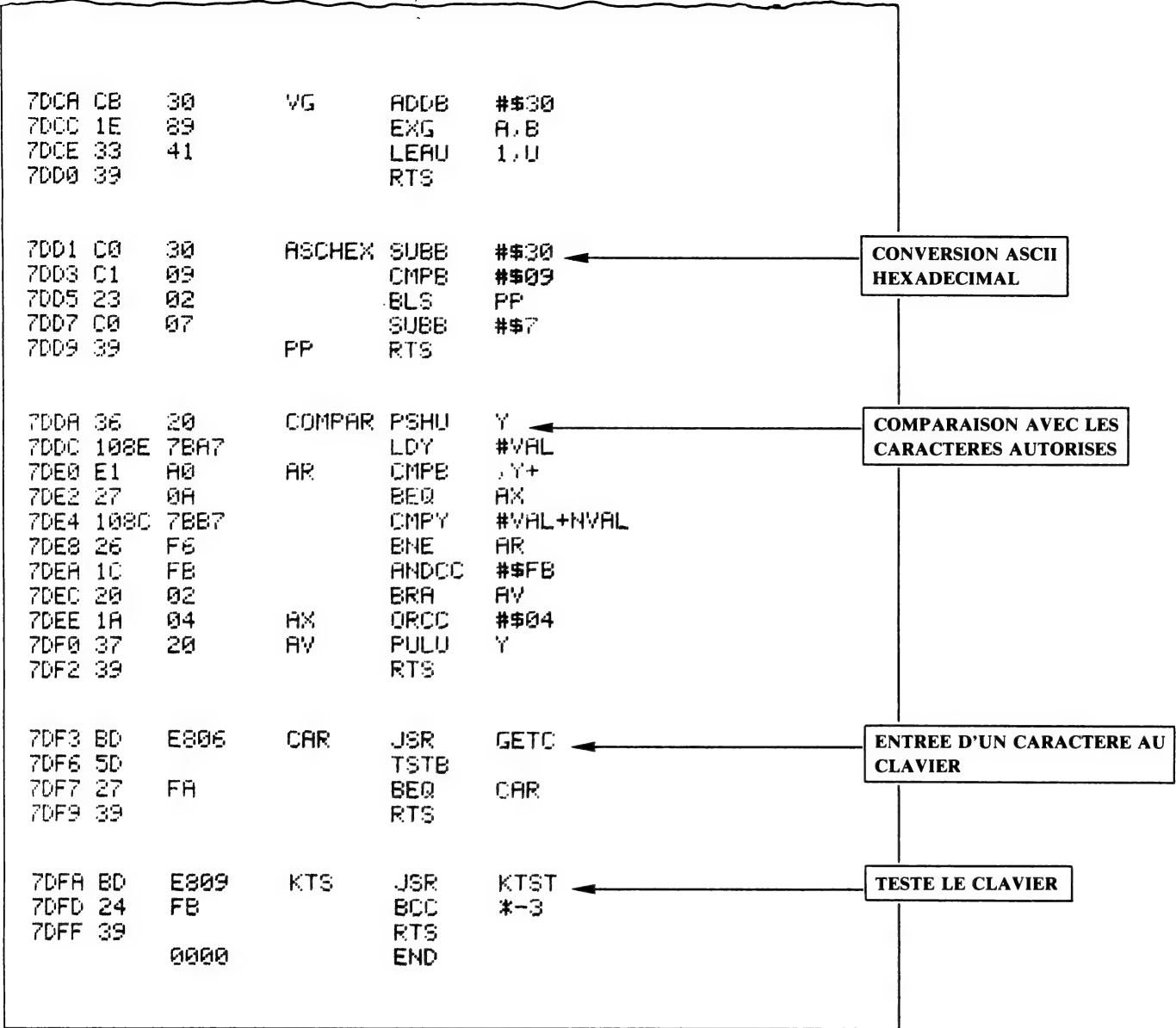
TRAIT

7C84	7F	7BB9	CLR	RTENU	← RESULTAT
7C87	06	04	LDB	##04	
7C89	F7	7BB8	STB	NDIG	
7C8C	86	1A	LDA	#26T	COLONNE
7C8E	06	12	LDB	#18T	LIGNE
7C90	36	06	BOUCLE PSHU	A,B	
7C92	BD	7D64	JSR	POUDC	
7C95	E6	42	LDB	2,U	
7C97	FB	7BB9	ADDB	RTENU	
7C9A	EB	46	ADDB	6,U	
7C9C	C1	10	CMPB	##10	
7C9E	24	05	BHS	RET	
7CA0	7F	7BB9	CLR	RTENU	
7CA3	20	06	BRA	FR	
7CA5	7F	7BB9	RET CLR	RTENU	
7CA8	7C	7BB9	INC	RTENU	
7CAB	BD	7DAE	PR JSR	HEXASC	
7CAE	1E	09	EXG	A,B	
7CB0	BD	E803	JSR	PUTC	
7CB3	7D	7BB9	TST	RTENU	
7CB6	27	02	BEQ	VB	
7CB8	8D	4F	BSR	PRTENU	
7CBA	37	06	VB PULU	A,B	
7CBC	8D	02	SUBA	#02	
7CBE	7A	7BB8	DEC	NDIG	
7CC1	27	0A	BEQ	SUITE	
7CC3	BD	7D1D	JSR	PAUSE	
7CC6	BD	7DFA	JSR	KTS	
7CC9	33	41	LEAU	1,U	
7CCB	20	C3	BRA	BOUCLE	
7CCD	BD	7DA1	SUITE JSR	TANOR	
7CD0	86	01	LDA	#01	← colonne
7CD2	06	18	LDB	#24T	ligne
7CD4	BD	7D72	JSR	PCUR	
7CD7	8E	7B18	LDX	#MESS0	
7CDA	BD	7D95	JSR	MESSAG	
7CDD	86	01	LDA	#01	← MESSAGE 2
7CDF	06	17	LDB	#23T	
7CE1	BD	7D72	JSR	PCUR	
7CE4	8E	7B62	LDX	#MESS2	
7CE7	BD	7D95	JSR	MESSAG	
7CEA	86	01	LDA	#01	← MESSAGE 3
7CEC	06	18	LDB	#24T	
7CEE	BD	7D72	JSR	PCUR	
7CF1	8E	7B83	LDX	#MESS3	
7CF4	BD	7D95	JSR	MESSAG	
7CF7	BD	7D1D	JSR	PAUSE	
7CFA	BD	7DF3	JSR	CAR	
7CFD	C1	0D	CMPB	#CR	
7CFF	1027	FEE4	LBEQ	NADD	
7D03	06	1E	LDB	##1E	en haut gauche
7D05	BD	E803	JSR	PUTC	
7D08	3F		SWI		

*****SOUS-PROGRAMMES*****

7D09	BD	7DA1	PRTENU	JSR	TANOR	IMPRESSION DE LA RETENUE
7D0C	37	06		PULU	A,B	
7D0E	36	06		PSHU	A,B	
7D10	06	08		LDB	#08T	
7D12	80	02		SUBA	#02	
7D14	BD	7D72		JSR	POUR	
7D17	06	31		LDB	#'1	
7D19	BD	E803		JSR	PUTC	
7D1C	39			RTS		
7D1D	36	10	PAUSE	PSHU	X	PAUSE
7D1F	8E	FFFF		LDX	#\$FFFF	
7D22	30	1F	ENC	LEAX	-1,X	
7D24	26	FC		BNE	ENC	
7D26	37	10		PULU	X	
7D28	39			RTS		
7D29	36	06	POFEN	PSHU	A,B	POSITIONNEMENT DE LA FENETRE COURANTE
7D2B	06	1F		LDB	#\$1F	
7D2D	BD	E803		JSR	PUTC	
7D30	E6	C4		LDB	,U	
7D32	54			LSRB		
7D33	54			LSRB		
7D34	54			LSRB		
7D35	54			LSRB		
7D36	C4	1F		ANDB	#\$1F	
7D38	CA	10		ORB	#\$10	
7D3A	BD	E803		JSR	PUTC	
7D3D	E6	C4		LDB	,U	
7D3F	C4	1F		ANDB	#\$1F	
7D41	CA	10		ORB	#\$10	
7D43	BD	E803		JSR	PUTC	
7D46	C6	1F		LDB	#\$1F	
7D48	BD	E803		JSR	PUTC	
7D4B	E6	41		LDB	1,U	
7D4D	54			LSRB		
7D4E	54			LSRB		
7D4F	54			LSRB		
7D50	54			LSRB		
7D51	C4	2F		ANDB	#\$2F	
7D53	CA	20		ORB	#\$20	
7D55	BD	E803		JSR	PUTC	
7D58	E6	41		LDB	1,U	
7D5A	C4	2F		ANDB	#\$2F	
7D5C	CA	20		ORB	#\$20	
7D5E	BD	E803		JSR	PUTC	
7D61	37	06		PULU	A,B	
7D63	39			RTS		

7D64	36	06	POUDC	PSHU	A, B	POSITIONNEMENT DU CURSEUR + double caractère
7D66	06	1B		LDB	##\$1B	
7D68	BD	E803		JSR	PUTC	
7D6B	06	4F		LDB	##\$4F	
7D6D	BD	E803		JSR	PUTC	
7D70	37	06		PULU	A, B	
7D72	36	06	PCUR	PSHU	A, B	POSITIONNEMENT DU CURSEUR
7D74	06	1F		LDB	##\$1F	
7D76	BD	E803		JSR	PUTC	
7D79	37	06		PULU	A, B	
7D7B	0B	40		ADDB	##\$40	
7D7D	BD	E803		JSR	PUTC	
7D80	1F	89		TFR	A, B	
7D82	0B	40		ADDB	##\$40	
7D84	BD	E803		JSR	PUTC	
7D87	39			RTS		
7D88	36	04	MATT	PSHU	B	MODIFICATION D'ATTRIBUTS
7D8A	06	1B		LDB	##\$1B	
7D8C	BD	E803		JSR	PUTC	
7D8F	37	04		PULU	B	
7D91	BD	E803		JSR	PUTC	
7D94	39			RTS		
7D95	E6	80	MESSAG	LDB	,X+	ECRITURE D'UN MESSAGE SUR L'ECRAN
7D97	C1	04		CMPB	##\$04	
7D99	27	05		BEQ	*+07	
7D9B	BD	E803		JSR	PUTC	
7D9E	20	F5		BRA	MESSAG	
7DA0	39			RTS		
7DA1	06	1B	TANOR	LDB	##\$1B	MISE EN TAILLÉ NORMALE
7DA3	BD	E803		JSR	PUTC	
7DA6	06	4C		LDB	##\$4C	
7DA8	BD	E803		JSR	PUTC	
7DAB	06	1F		LDB	##\$1F	
7DAD	39			RTS		
7DAE	36	04	HEXASC	PSHU	B	CONVERSION HEXADÉCIMAL ASCII
7DB0	54			LSRB		
7DB1	54			LSRB		
7DB2	54			LSRB		
7DB3	54			LSRB		
7DB4	04	0F		ANDB	##\$F	
7DB6	01	09		CMPB	##\$09	
7DB8	23	02		BLS	IG	
7DBA	0B	07		ADDB	#07	
7DBC	0B	30	IG	ADDB	##\$30	
7DBE	1F	98		TFR	B, A	
7DC0	E6	04		LDB	,U	
7DC2	04	0F		ANDB	##\$F	
7DC4	01	09		CMPB	##\$09	
7DC6	23	02		BLS	VG	
7DC8	0B	07		ADDB	#07	



SOUSTRACTION HEXADÉCIMALE SUR 4 DIGITS

* BG **SOUSTRACTION HEXADÉCIMALE *** RW*
***** sur 4 digits *****

		E809	KTST	EQU	\$E809	
		E833	CHPL	EQU	\$E833	
		E80C	DRAW	EQU	\$E80C	
		6041	CHDRAW	EQU	\$6041	
		E806	GETC	EQU	\$E806	
		E803	PUTC	EQU	\$E803	
		000D	CR	EQU	\$0D	
7B00	2A	53	4F	55	MESST	FCC / *SOUSTRACTION HEXADÉCIM/
7B04	53	54	52	41		
7B08	43	54	49	4F		
7B0C	4E	20	48	45		
7B10	58	41	44	45		
7B14	43	49	4D			
7B17	41	4C	45	2A		
7B1B	04				FCC /ALE*/	
7B1C	20	20	20	20	FCB \$04	delimiteur
7B20	20	20	20	20	MES00 FCC /	/
7B24	20	20	20	20		
7B28	20	20	20	20		
7B2C	20	20	20	20		
7B30	20	20	20	20		
7B34	20	20	20	20	FCC /	/
7B38	20	20	20	20		
7B3C	20	20	20	20		
7B40	20	20	20	20		
7B44	20					
7B45	04				FCB \$04	
7B46	46	72	61	70	MES11 FCC	/Frappez un caractere HEX/
7B4A	70	65	7A	20		
7B4E	75	6E	20	63		
7B52	61	72	61	63		
7B56	74	65	72	65		
7B5A	20	48	45	58		
7B5E	41	44	45	43	FCC /ADECIMAL/	
7B62	49	4D	41	4C		
7B66	04				FCB \$04	

```

7B67 4E 6F 75 76 MESS2 FCC /Nouvelle soustraction ->/
7B68 65 6C 6C 65
7B6F 20 73 6F 75
7B73 73 74 72 61
7B77 63 74 69 6F
7B7B 6E 20 2D 3E
7B7F 20 45 4E 54 FCC / ENTREE /
7B83 52 45 45 20
7B87 04 FCB $04
7B88 52 65 74 6F MESS3 FCC /Retour au moniteur ---> /
7B8C 75 72 20 61
7B90 75 20 6D 6F
7B94 6E 69 74 65
7B98 75 72 20 2D
7B9C 2D 2D 3E 20
7BA0 20 75 6E 65 FCC / une TOUCHE/
7BA4 20 54 4F 55
7BA8 43 48 45
7BAB 04 FCB $04
          0010 NVAL EQU 16T
7BAC 30 31 32 33 VAL FCC /0123456789ABCDEF/
7BB0 34 35 36 37
7BB4 38 39 41 42
7BB8 43 44 45 46
7BBC 0D FCB CR
7BD0 NDIG RMB 1 nombre de digit
7BDE RTENU RMB 1 RETENUE
7BDF CE C000 SOUS LDU #ENDMEM

```

```

7BC2 C6 62 LDB #$62 tour vert ← Couleur du tour
7BC4 BD 7D8D JSR MATT

```

```

7BC7 C6 6B LDB #$6B ← Mode page
7BC9 BD 7D8D JSR MATT

```

```

7BCD 86 24 LDA #$24 LIGNE BASSE ← Positionnement de la fenêtre sur la
7BCE C6 00 LDB #$00 LIGNE HAUTE totalité de l'écran
7BD0 BD 7D2E JSR POFEN

```

```

7BD3 C6 0C LDB #$0C ← Effacement de la totalité de l'écran
7BD5 BD E803 JSR PUTC

```

```

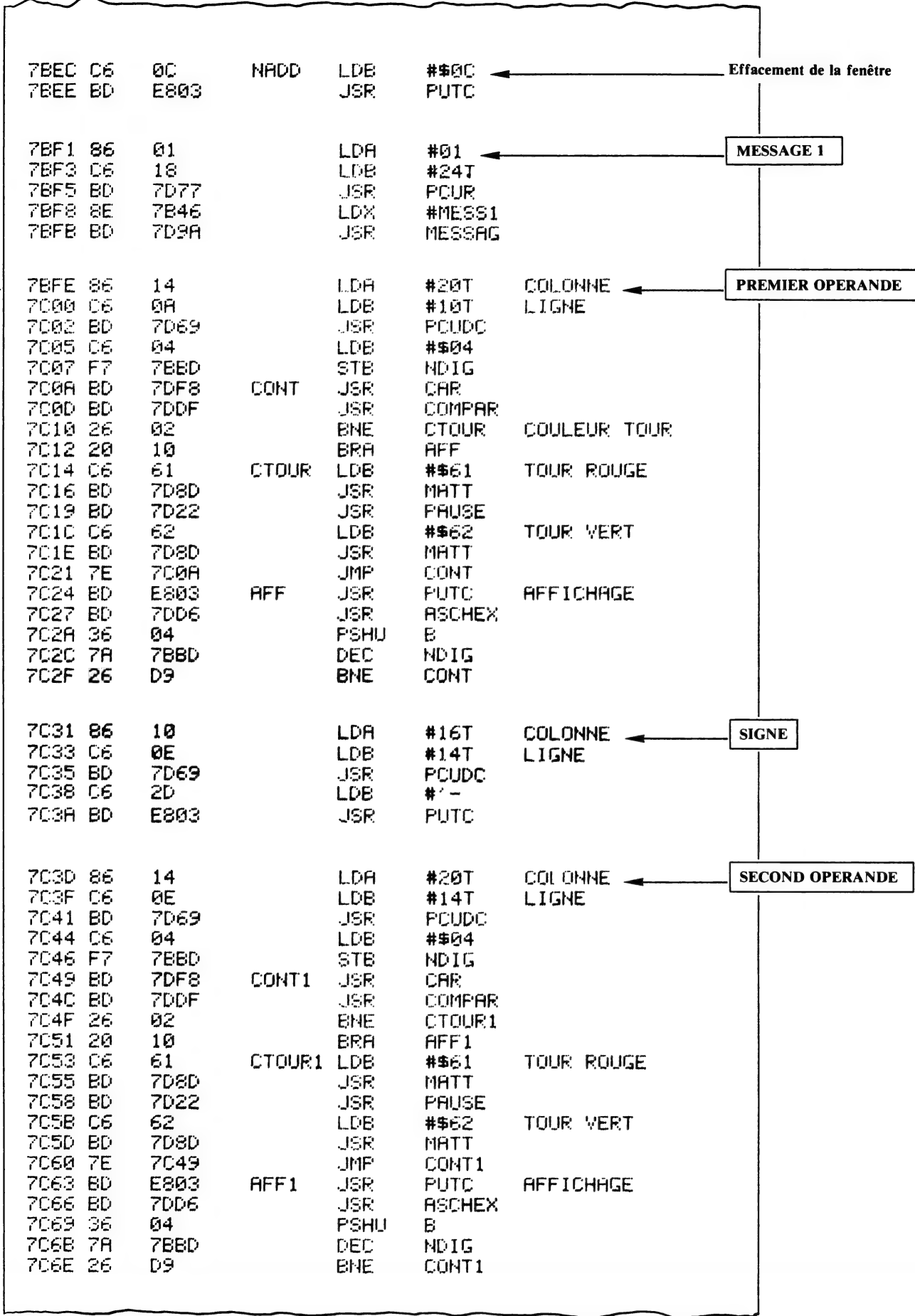
7BD8 86 0A LDA #10T colonne ← MESSAGE titre
7BDA C6 00 LDB #00T ligne
7BDC BD 7D77 JSR POUR
7BDF 8E 7B00 LDX #MESS1
7BE2 BD 7D9A JSR MESSAG

```

```

7BE5 86 24 LDA #$24 LIGNE BASSE ← Positionnement de la fenêtre sur la
7BE7 C6 06 LDB #$06 LIGNE HAUTE partie de l'écran modifiable
7BE9 BD 7D2E JSR POFEN

```



7C70	8E	0010	LDX	#16T
7C73	108E	0010	LDY	#16T
7C77	06	2D	LDB	#'-
7C79	F7	6041	STB	CHDRAW
7C7C	BD	E833	JSR	CHPL
7C7F	8E	0010	LDX	#28T
7C82	108E	0010	LDY	#16T
7C86	BD	E80C	JSR	DRAW

TRAIT

7C89	7F	7BBE	CLR	RTENU
7C8C	06	04	LDB	#04
7C8E	F7	7BB0	STB	NDIG
7C91	86	1A	LDA	#26T
7C93	06	12	LDB	#18T
7C95	36	06	BOUCLE	PSHU
7C97	BD	7D69	JSR	PCUDC
7C9A	E6	46	LDB	6,U
7C9C	F0	7BBE	SUBB	RTENU
7C9F	E0	42	SUBB	2,U
7CA1	01	10	CMPE	#10
7CA3	24	05	BHS	RET
7CA5	7F	7BBE	CLR	RTENU
7CA8	20	06	BRA	PR
7CAA	7F	7BBE	RET	CLR
7CAD	7C	7BBE	INC	RTENU
7CB0	BD	7DB3	PR	JSR
7CB3	1E	89	EXG	A,B
7CB5	BD	E803	JSR	PUTC
7CB8	7D	7BBE	TST	RTENU
7CBB	27	02	BEQ	VB
7CBD	8D	4F	BSR	PRTENU
7CBF	37	06	VB	PULU
7CC1	80	02	SUBA	#02
7CC3	7A	7BB0	DEC	NDIG
7CC6	27	0A	BEQ	SUITE
7CC8	BD	7D22	JSR	PAUSE
7CCB	BD	7DFF	JSR	KTS
7CCE	33	41	LEAU	1,U
7CD0	20	C3	BRA	BOUCLE
7CD2	BD	7DA6	SUITE	JSR

RESULTAT

COLONNE
LIGNE

MESSAGE 0

colonne
ligne

MESSAGE 2

70EF	86	01	LDA	#01	
70F1	06	18	LDB	#24T	
70F3	BD	7D77	JSR	PCUR	
70F6	8E	7B88	LDX	#MESS3	
70F9	BD	7D9A	JSR	MESSAG	
70FC	BD	7D22	JSR	PAUSE	
70FF	BD	7DF8	JSR	CAR	
7D02	C1	0D	CMPB	#CR	
7D04	1027	FEE4	LBEQ	NADD	
7D08	06	1E	LDB	##1E	en haut gauche
7D0A	BD	E803	JSR	PUTC	
7D0D	3F		SWI		

MESSAGE 3

*****SOUS-PROGRAMMES*****

7D0E	BD	7DA6	PRTENU	JSR	TANOR
7D11	37	06		PULU	A,B
7D13	36	06		PSHU	A,B
7D15	06	08		LDB	#08T
7D17	80	02		SUBA	#02
7D19	BD	7D77		JSR	PCUR
7D1C	06	31		LDB	#'1
7D1E	BD	E803		JSR	PUTC
7D21	39			RTS	

IMPRESSION DE LA RETENUE

7D22	36	10	PAUSE	PSHU	X
7D24	8E	FFFF		LDX	##FFFF
7D27	30	1F	ENC	LEAX	-1,X
7D29	26	FC		BNE	ENC
7D2B	37	10		PULU	X
7D2D	39			RTS	

PAUSE

7D2E	36	06	POFEN	PSHU	A,B
7D30	06	1F		LDB	##1F
7D32	BD	E803		JSR	PUTC
7D35	E6	C4		LDB	,U
7D37	54			LSRB	
7D38	54			LSRB	
7D39	54			LSRB	
7D3A	54			LSRB	
7D3B	C4	1F		ANDB	##1F
7D3D	CA	10		ORB	##10
7D3F	BD	E803		JSR	PUTC
7D42	E6	C4		LDB	,U
7D44	C4	1F		ANDB	##1F
7D46	CA	10		ORB	##10
7D48	BD	E803		JSR	PUTC
7D4B	C6	1F		LDB	##1F
7D4D	BD	E803		JSR	PUTC
7D50	E6	41		LDB	,U
7D52	54			LSRB	
7D53	54			LSRB	
7D54	54			LSRB	
7D55	54			LSRB	

POSITIONNEMENT DE LA
FENETRE COURANTE


```

7D56 C4 2F      ANDB  #$2F
7D58 CA 20      ORB   #$20
7D5A BD E803    JSR   PUTC
7D5D E6 41      LDB   1,U
7D5F C4 2F      ANDB  #$2F
7D61 CA 20      ORB   #$20
7D63 BD E803    JSR   PUTC
7D66 37 06      PULU  A,B
7D68 39

```

```

7D69 36 06      PCUDD PSHU  A,B ←
7D6B C6 1B      LDB   #$1B
7D6D BD E803    JSR   PUTC
7D70 C6 4F      LDB   #$4F
7D72 BD E803    JSR   PUTC
7D75 37 06      PULU  A,B

```

POSITIONNEMENT DU
CURSEUR + double caractère

```

7D77 36 06      PCUR  PSHU  A,B ←
7D79 C6 1F      LDB   #$1F
7D7B BD E803    JSR   PUTC
7D7E 37 06      PULU  A,B
7D80 C6 40      ADDB  #$40
7D82 BD E803    JSR   PUTC
7D85 1F 89      TFR   A,B
7D87 C6 40      ADDB  #$40
7D89 BD E803    JSR   PUTC
7D8C 39

```

POSITIONNEMENT DU
CURSEUR

```

7D8D 36 04      MATT  PSHU  B ←
7D8F C6 1B      LDB   #$1B
7D91 BD E803    JSR   PUTC
7D94 37 04      PULU  B
7D96 BD E803    JSR   PUTC
7D99 39

```

MODIFICATION D'ATTRIBUTS

```

7D9A E6 80      MESSAG LDB   ,X+ ←
7D9C C1 04      CMPB  #$04
7D9E 27 05      BEQ   *+07
7DA0 BD E803    JSR   PUTC
7DA3 20 F5      BRA   MESSAG
7DA5 39

```

ECRITURE D'UN MESSAGE SUR
L'ECRAN

```

7DA6 C6 1B      TANOR  LDB   #$1B ←
7DA8 BD E803    JSR   PUTC
7DAB C6 4C      LDB   #$4C
7DAD BD E803    JSR   PUTC
7DB0 C6 1F      LDB   #$1F
7DB2 39

```

MISE EN TAILLE NORMALE

7DB3	36	04	HEXASC	PSHU	B
7DB5	54			LSRB	
7DB6	54			LSRB	
7DB7	54			LSRB	
7DB8	54			LSRB	
7DB9	C4	0F		ANDB	#\$F
7DBB	C1	09		CMPB	#\$09
7DBD	23	02		BLS	IG
7DBF	CB	07		ADDB	#07
7DC1	CB	30	IG	ADDB	#\$30
7DC3	1F	90		TFR	B,A
7DC5	E6	C4		LDB	,U
7DC7	C4	0F		ANDB	#\$F
7DC9	C1	09		CMPB	#\$09
7DCB	23	02		BLS	VG
7DCD	CB	07		ADDB	#07
7DCF	CB	30	VG	ADDB	#\$30
7DD1	1E	89		EXG	A,B
7DD3	33	41		LEAU	1,U
7DD5	39			RTS	

CONVERSION HEXADÉCIMAL
ASCII

7DD6	C0	30	ASCHEX	SUBB	#\$30
7DD8	C1	09		CMPB	#\$09
7DDA	23	02		BLS	PP
7DDC	C0	07		SUBB	#\$7
7DDE	39		PP	RTS	

CONVERSION ASCII
HEXADÉCIMAL

7DDF	36	20	COMPAR	PSHU	Y
7DE1	102E	7BAC		LDY	#VAL
7DE5	E1	A0	AR	CMPB	,Y+
7DE7	27	0A		BEQ	AX
7DE9	108C	7BBC		CMPLY	#VAL+NVAL
7DED	26	F6		BNE	AR
7DEF	1C	FB		ANDCC	#\$FB
7DF1	20	02		BRA	AV
7DF3	1A	04	AX	ORCC	#\$04
7DF5	37	20	AV	PULU	Y
7DF7	39			RTS	

COMPARAISON AVEC LES
CARACTERES AUTORISES

7DF8	BD	E806	CAR	JSR	GETC
7DFB	5D			TSTB	
7DFC	27	FA		BEQ	CAR
7DFE	39			RTS	

ENTREE D'UN CARACTERE AU
CLAVIER

7DFF	BD	E809	KTS	JSR	KTST
7E02	24	FB		BCC	*-3
7E04	39			RTS	
		0000		END	

TESTE LE CLAVIER

4

MULTIPLICATION DÉCIMALE DE 2 NOMBRES DE 1 CHIFFRE

BUT : *Ecrire un programme à partir d'un algorithme et d'un organigramme.*

INSTRUCTIONS UTILISEES : *LDA, STA, ADDA, DEC, BNE.*

ADRESSAGE : *Etendu.*

Problème : Ecrire un programme réalisant la multiplication décimale de 2 nombres.

Question : Qu'est-ce qu'une multiplication ?

Réponse : La multiplication de 2 nombres est une suite d'additions, c'est-à-dire que si j'écris :

$$5 * 4$$

cela équivaut à :

$$5 + 5 + 5 + 5$$

soit 20.

Ce qui s'écrit en simplifiant et en généralisant :

$$P = a * b = \sum_{i=1}^b a$$

où a et b représentent les 2 nombres, P leur produit ; le signe Σ qu'on lit : somme de 1 à b de a, indique que l'on doit effectuer la somme (addition) de b termes (nombres) égaux à a.

Cette formule exprime l'**algorithme** (méthode de travail) que nous adoptons pour résoudre le problème posé.

Que devons-nous faire pour que notre ordinateur sache multiplier 2 nombres — aucun microprocesseur 8 bits ne peut le faire en une instruction.

Il faut :

- 1 donner a et b
- 2 mettre P à 0 (zéro)
- 3 ajouter à P, et ranger la nouvelle valeur de P, ce qui s'écrit :

$$P = P + a$$

- 4 faire $b = b - 1$, soit décompter le nombre d'opérations effectuées en 3
- 5 si b n'est pas nul ($b \neq 0$) on retourne à la ligne 3
- 6 si b est nul le travail est terminé.

Ces diverses opérations à effectuer dans un ordre bien précis conduisent au programme écrit en BASIC, suivant :

```

10 INPUT "A=";A:INPUT"B=",B
20 LET P=0
30 P=P+A
40 B=B-1
50 IF B>0 THEN GOTO 30
60 PRINT "P=";P

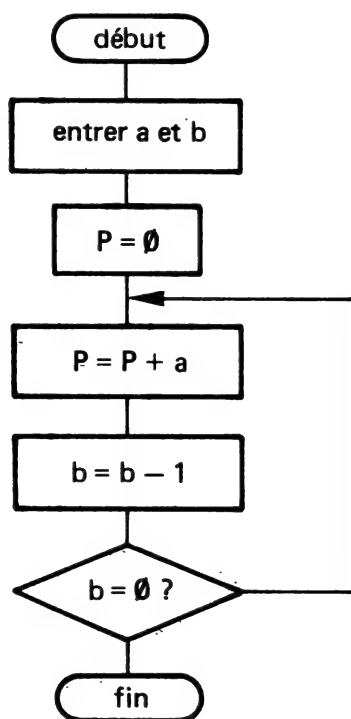
```

Ce programme est valable pour tout ordinateur « comprenant » le BASIC. Il ne précise pas la taille — nombre de chiffres — des nombres, parfois P sera exprimé en écriture « scientifique ».

Ce programme écrit sur votre ordinateur THOMSON, vous donnera le résultat de :

- 99 * 99 au bout de 1,6 secondes
- 999 * 999 au bout de 13 secondes !
- 9999 * 9999 au bout de... 2 minutes et 7 secondes !!!.

La série d'opérations à effectuer est exprimée de manière simplifiée et universelle sous forme de l'**organigramme** suivant :



En **assembleur** il faut :

- fixer la taille des nombres
- savoir où ranger a, b et P

Le 6809 est un microprocesseur 8 bits, il peut donc traiter simplement des nombres de 2 chiffres — il faut 4 bits par chiffre — aussi en appliquant le principe énoncé plus haut nous commencerons par le produit de 2 nombres de **un** chiffre.

Le rangement de a, b et P peut s'effectuer dans des registres ou des cases mémoires, cela dépend du processeur. Avec 8 bits on dispose généralement de 256 instructions, les constructeurs choisissent entre un jeu de registres d'usage général ou (souvent exclusif !) des modes d'adressage multiples.

Le 6809 entre dans la deuxième catégorie, il dispose de 2 registres 8 bits A et B qui sont des **accumulateurs** c'est-à-dire que les opérations arithmétiques ou logiques modifient leurs contenus, il **faut** donc utiliser les cases mémoires. Ainsi a, b et P seront « stockés » en mémoire après être passés par l'un des accumulateurs qui ne pourra être « chargé », dans l'état actuel de nos connaissances, que par programme (pour modifier a ou b il faudra modifier le programme). Il s'agit d'un petit handicap que nous comblerons rapidement...

La première instruction sera donc un **chargement immédiat** par exemple de A soit, si a vaut 3 :

```
LDA # $03
```

Le signe # signifie, ici, immédiat c'est-à-dire que la quantité qui suit sera mise dans A. Ce que l'on écrit :

```
03 → (A)
```

Le signe \$ signifie que la quantité qui suit est en *hexadécimal* et ne sera **pas** modifiée lors de l'assemblage.

Le contenu de l'accumulateur doit ensuite être *stocké* en mémoire. Nous choisissons comme mode d'adressage le plus simple, à savoir le *mode étendu* dans lequel l'adresse de la case mémoire est donnée en *clair* à l'aide d'un *mot* de 16 bits. Nous disposons de plusieurs « kilo-octets » à partir de l'adresse \$6100 (hexadécimal !) mais l'écriture de notre programme en mnémoniques avec commentaires prend de la place (un signe = un octet), aussi très prudemment fixons-nous la première case de stockage à l'adresse \$7100.

Le début du programme devient donc, avec a = 3 et b = 2 :

```
LDA # $03
STA $7100
LDA # $02
STA $7101
```

La mise à zéro du contenu de la case appelée P est effectuée à l'aide de l'instruction CLR (*clear*).

```
CLR $7102
```

Le problème le plus délicat à résoudre est : $P = P + a$.

En effet les opérations arithmétiques « passent » par l'accumulateur — **aucun** microprocesseur n'autorise l'addition directe entre 2 cases mémoires. Il faut donc mettre a (ou P) dans A (ou B), l'ajouter à P (ou a) et ranger (stocker) le résultat à la place de l'ancienne valeur de P. Ce que nous réalisons ainsi :

```
LDA $7102 (A) ← P
ADDA $7100 (A) = (A) + a
STA $7102
```

Par contre nous pouvons retirer 1 (décrémenter) du contenu d'une case mémoire **sans** passer par un accumulateur.

```
DEC $7101
```

Cette opération affecte les *flags* (indicateurs, drapeaux), en particulier celui qui indique que le résultat d'une opération est nul, le flag Z. Les instructions du style GOTO (aller à...) sont des **branchements**. Les mnémoniques correspondant aux branchements **conditionnés** sur la valeur de Z sont :

```
BEQ (equal, égal) si le résultat est nul (Z = 1)
BNE (non égal) si le résultat est différent de 0
```

La destination peut être donnée à l'aide d'une adresse (mot de 16 bits) ou à l'aide d'un symbole ou *étiquette* (*label*).

Dans le premier cas **il faut** connaître les adresses de chaque instruction donc le nombre d'octets qu'elle requiert... travail nécessaire si l'on utilise directement le langage machine.

Notre logiciel d'assemblage nous permet l'emploi d'étiquettes. Il s'agit de mots, plus ou moins abrégés ou mal orthographiés qui indiquent des points particuliers dans le programme ou, comme nous le verrons plus loin, des sous-programmes, des données...

Cela est possible car le logiciel d'assemblage effectue son travail en « deux passes ». A la première il *code* les instructions et repère les étiquettes (point d'entrée ou définition et point(s) de sortie) dont les adresses ne peuvent être fixées que lors de la deuxième passe.

Le nombre de caractères des labels est, ici, limité à 6.

Nous devons nous brancher à l'instruction LDA \$7102, que nous baptisons BOUCLE soit :

```
BOUCLE LDA $7102
```

Le mot BOUCLE est écrit dans la colonne (*field*) label.
L'instruction de branchement est alors :

```
BNE BOUCLE
```

Comment indiquer au processeur que le travail est terminé ? dans la majorité des systèmes en *rendant la main au moniteur*.

Le moniteur est un ensemble de programmes nous permettant de « lancer » le nôtre mais surtout de le mettre au point grâce :

- à l'examen (modification) des contenus des registres
- à l'examen (modification) des contenus des cases mémoires
- à l'exécution en pas-à-pas (instruction par instruction)

Pour rendre la main au moniteur nous utilisons, ici, l'instruction

```
SWI
```

Afin que le programme qui va effectuer l'assemblage de notre **programme source** le fasse sans nous adresser des messages d'erreur nous devons indiquer par une étiquette le début du programme et la fin. Pour cette dernière on utilise la *directive* END.

Nous désirons que le **programme objet** commence à une adresse précise, nous l'indiquons à l'aide de la directive ORG.

Nous obtenons le listage suivant :

	ORG	\$7000
DEBUT	LDA	##03
	STA	\$7100
	LDA	##02
	STA	\$7101
	CLR	\$7102
BOUCLE	LDA	\$7102
	ADDA	\$7100
	STA	\$7102
	DEC	\$7101
	BNE	BOUCLE
	SWI	
	END	DEBUT

Et l'assemblage nous fournit :

7000			ORG	\$7000
7000	86	03	DEBUT	LDA #03
7002	B7	7100		STA \$7100
7005	86	02		LDA #02
7007	B7	7101		STA \$7101
700A	7F	7102		CLR \$7102
700D	B6	7102	BOUCLE	LDA \$7102
7010	BB	7100		ADDA \$7100
7013	B7	7102		STA \$7102
7016	7A	7101		DEC \$7101
7019	26	F2		BNE BOUCLE
701B	3F			SWI
		7000	END	DEBUT

EXECUTION ET MISE AU POINT

L'assemblage ayant été exécuté sans déceler d'erreur ! nous pouvons passer à l'exécution en appelant le MONITEUR et en « lançant » le programme par un :

GDEBUT plein d'angoisse...!!

La réponse ne tarde pas :

8 BRK @ DEBUT

Ouf ! le système n'est pas parti dans l'espace temps, ça arrive...

Pour obtenir le résultat il faut examiner le contenu de la case mémoire d'adresse 7102 après avoir demandé une réponse *en hexadécimal* par :

#N

suivi de

#7102/

la réponse est immédiate :

#7102/ 6

ÇA MARCHE !!!... ?

En êtes-vous sûr !... si votre réponse est OUI, relisez la première page du livre... ça marche dans **un** cas, voyons d'autres produits. Pour cela il faut revenir au programme source et corriger les nombres a et b, ou puisque nous avons le listage avec les codes machines et les adresses, nous pouvons modifier le programme objet — **attention** le programme source n'est pas modifié — pour cela on appelle les cases mémoires d'adresse 7001 (a) et 7006 (b) et on essaie $5 * 3$, soit :

#7001/ 3 5
#7006/ 2 3

On lance le programme comme précédemment, la réponse dans ce cas est...

#7102/ 0F

Notre programme donne donc $5 * 3 = F$!!

Pour comprendre d'où vient « l'erreur » nous exécutons le travail en *pas-à-pas* à l'aide de la commande T. Mais pour ce faire il faut d'abord demander un *point d'arrêt* qui rend la main au moniteur sans avoir à insérer des SWI. On tape donc :

#KDEBUT

Puis

#GDEBUT

la réponse est

Ø BRK @ DEBUT

Nous commandons alors le pas-à-pas, faisant apparaître le contenu des registres PC,A,B,DP,CC,X,Y,U et S ce qui nous permet de suivre les opérations effectuées, d'autant que le système nous indique, en mnémonique, quelle instruction il va exécuter. Ainsi :

T
7002 05 00 00 80 0000 0000 0000 6303
7002 STA >7100

indique que A a été chargé à 5 et que (A) va être stocké en 7100.

L'exécution complète en pas-à-pas est un peu longue mais elle permet de comprendre comment travaille notre 6809.

On trouve en 7013 :

7013 05 00 00 80 0000 0000 0000 6303
7013 STA >7102

Puis de nouveau en 7013 :

7013 0A 00 00 80 0000 0000 0000 6303
7013 STA >7102

et une dernière fois en 7013 :

7013 0F 00 00 80 0000 0000 0000 6303
7013 STA >7102

Puisque en 7019, après DEC >7101 on a :

7019 0F 00 00 84 0000 0000 0000 6303
7019 BNE BOUCLE

qui est suivi de :

701B 0F 00 00 84 0000 0000 0000 6303
701B SWI

indiquant que le travail est terminé. En 7019 le flag Z est passé à 1 car le contenu de la case d'adresse 7101 venait de passer à 0 (Z est le bit 2 de CC :EFHINZVC)

Notre problème commence donc à 5 + 5 = A, vérifions :

$$\begin{array}{r} 5 = 0101 \\ + 5 = 0101 \\ \hline 10 = 1010 ? \end{array}$$

Le nombre de gauche est en décimal, celui de droite son expression en binaire que la machine nous transmet en *hexadécimal* soit A, le phénomène se reproduit avec A + 5 qui donne F

$$\begin{array}{rcl} 5 & = & 0101 \\ + A & = & 1010 \\ \hline F & = & 1111 ? \end{array}$$

Si nous désirons un programme qui réalise la multiplication *décimale* il faut **traduire** A en 10, F en 15... ce qui exige un « *programme* » si le travail est effectué avant l'envoi du résultat à l'opérateur (voir programmes 17 à 20).

Mais, dans le cas de l'addition, nous disposons d'une instruction, qui, associée à *ADDA* ou *ADCA*, convertit le résultat en décimal *si* les nombres additionnés étaient **décimaux**. Il s'agit de *DAA* (*Decimal Adjust for Addition*) qui opère sur le contenu de l'*accumulateur* A.

Cette instruction ajoute 6 à un quartet (4 bits ou nibble) s'il est supérieur à 9 ou si la retenue correspondante (C pour le quartet de poids fort ou H pour le quartet de poids faible) vaut 1. Ainsi nous aurons :

$$\begin{array}{rcl} 05 & = & 0000 \quad 0101 \\ + 05 & = & 0000 \quad 0101 \\ \hline 0A & = & 0000 \quad 1010 \\ + 06 & = & 0000 \quad 0110 \\ \hline 0001 & 0000 & = 10 \end{array}$$

A > 9 donc

nous « lisons » 10 (dix) — remarquez les guillemets !!! — si nous ajoutons 8 à 8 nous aurons :

$$\begin{array}{rcl} 08 & = & 0000 \quad 1000 \\ + 08 & = & 0000 \quad 1000 \\ \hline 10 & = & 0001 \quad 0000 \\ + 06 & = & 0000 \quad 0110 \\ \hline 0001 & 0110 & = 16 \end{array}$$

ici H = 1 donc

nous « lisons » 16 (seize).

Après *DAA* le contenu de A peut donc être augmenté de 00, 06, 60 ou 66 **mais DAA n'est utile qu'après ADDA ou ADCA** puisque la valeur **instantanée** des flags C et H est prise en compte.

Le nouveau listage (après assemblage) est :

7000			ORG	\$7000
7000	86	05	DEBUT	LDA #05
7002	B7	7100		STA \$7100
7005	86	03		LDA #03
7007	B7	7101		STA \$7101
700A	7F	7102		CLR \$7102
700D	B6	7102	BOUCLE	LDA \$7102
7010	BB	7100		ADDA \$7100
7013	19			DAA
7014	B7	7102		STA \$7102
7017	7A	7101		DEC \$7101
701A	26	F1		BNE BOUCLE
701C	3F			SWI
		7000	END	DEBUT

Le résultat obtenu pour $5 * 3$ est bien dans ce cas, 15 ; de même nous obtenons 15 pour $3 * 5$ (commutativité de la multiplication).

L'essai en pas-à-pas vous permettra d'observer le travail de l'instruction DAA qui, dans le cas de $5 * 3$, traduit 0A en 10, dans le cas de $8 * 6$, par exemple, traduira successivement :

10	en 16	$8 * 2$
1E	en 24	$8 * 2 + 8 = 8 * 3$
2C	en 32	$8 * 4$
3A	en 40	$8 * 5$
enfin	48	en 48 $8 * 6$

Le programme que nous venons de mettre au point répond bien au problème posé puisque le résultat est correct pour tout produit de 2 nombres de 1 chiffre différent de 0.

Mais qu'en est-il de $15 * 3$?

Avec surprise nous constatons que le résultat est bien 45.

Le programme « marcherait-il » quand l'un des nombres est de 2 chiffres ? (la question ne se pose pas pour 2 nombres de 2 chiffres puisque dans ce cas le résultat peut en compter 4).

L'essai de $3 * 15$ qui donne ... 63 nous conduit au problème suivant...

5

CONVERSION DCB-HEXADÉCIMAL POUR UN NOMBRE DE DEUX CHIFFRES

BUT : Etude des masques et des rotations, préparation à la multiplication de nombres de 2 chiffres.

INSTRUCTIONS UTILISEES : *ANDA, LSRA.*

ADRESSAGE : *Etendu.*

Nos essais précédents de $15 * 3$ et $3 * 15$, ont donné respectivement 45 et... 63 !... d'où l'intérêt des tests multiples lors de la mise au point d'un programme. Si le premier résultat est correct, pourquoi le deuxième ne l'est-il pas ?

Il est évident que nous obtenons $3 * 21$, pourquoi ? La démarche la plus simple et la plus rapide est l'analyse du programme en pas-à-pas. Pour être plus efficace, nous mettrons un point d'arrêt à DEC \$7101, car l'addition est correcte, à l'aide de :

#K7017

Le déroulement du programme s'arrêtera à chaque décrémentation, il faudra examiner la case d'adresse 7101. Nous avons successivement après « lancement » du programme (le point d'arrêt en DEBUT a disparu lors de l'assemblage) :

```
#GDEBUT
  0 BRK @ 7017
#7101/    14
#C                (continue !)
  0 BRK @ 7017
#7101    13
,,
,,
,,
,,
,,
  0 BRK @ 7017
#7101/    10      Pour l'instant cela marche correctement
#C
  0 BRK @ 7017
#7101/    0F      et non 09! où est « l'erreur » ?
```

Pour retrancher 1 à un nombre on ajoute le complément à 2 de 1 soit FF (1111 1111), donc $10 - 1$ est égal à :

$$\begin{array}{r} 10\ 0001\ 0000 \\ + (-1) = 1111\ 1111 \\ \hline 0000\ 1111 \end{array}$$

Remarque : La décrémentation (comme l'incrément) n'affecte pas le carry.

Nous constatons que pour aller de 10 à 09, il faut passer par 0F, 0E, 0D, 0C, 0B et 0A, soit effectuer 6 boucles supplémentaires. Vous pourrez vérifier qu'il en est de même à chaque dizaine.

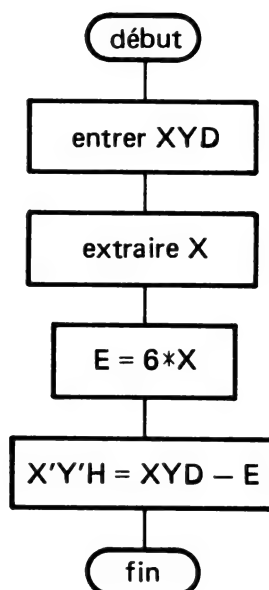
Pour avoir un programme qui « marche » nous pourrions donner 0F à la machine, à la place de 15, mais que ferez-vous pour 99*99 ?

Il faut donc écrire un programme de conversion qui traduise le nombre de boucles à effectuer en hexadécimal.

Nous venons de voir, grâce au pas-à-pas, que nous effectuions 6 boucles excédentaires par dizaine que compte le multiplicateur. Nous en tirons la formule suivante :

$$X'Y'H = XYD - 6 * X$$

où X',Y' sont les digits du nombre en hexadécimal, et X,Y les chiffres (compris entre 0 et 9) du nombre décimal. Cette formule conduit à l'organigramme suivant :



Comme on peut le constater il y a 2 problèmes :

- extraire X
- $E = 6 * X$

Extraire X ? Nous pouvons, logiquement le faire grâce à un ET de XYD avec F0 (1111 0000) puisque 1 ET x donne x, et que 0 ET x donne 0 soit :

xxxx yyyy où xxxx représente X, et yyyy, Y
 ET 1111 0000
 xxxx 0000

Nous avons utilisé un *masque* qui met quelques bits à 0.

Pour obtenir 0X il faut effectuer 4 rotations (à droite ou à gauche selon notre humeur du moment puisque nous aurons dans les 2 cas 0X comme résultat). Mais nous pouvons avec le 6809 effectuer des divisions par 2 **non** signées (un 0 est mis en bit de poids fort à chaque rotation), solution que nous adoptons :

```

LDA  #$15 Pour essai
LSRA  XY/2
LSRA  XY/4
LSRA  XY/8
LSRA  XY/16 (A)=0X
  
```

Multiplier X par 6 ! Nous pourrions évidemment utiliser le principe du programme précédent, mais si on remarque que :

6 = 4 + 2

avec

4 = 2 * 2 ou 8 / 2

et que :

XY ET F0 = 0X * 16

on arrive à la séquence suivante :

```
LDA    #$15    Pour essai
ANDA   #$F0    (A) = X0
LSRA                   (A) = X * 8
LSRA                   (A) = X * 4
STA     $7103   tampon
LSRA                   (A) = X * 2
ADDA    $7103   (A) = X * 6
```

nous avons besoin d'une case « tampon » car ADDA B — addition de (A) à (B) n'existe pas.

Il suffit, ensuite de rappeler (non prévu dans les lignes précédentes) le nombre XYD, de lui retrancher le contenu de l'accumulateur pour obtenir le résultat désiré, ce qui donne le listage suivant :

```

                                *CONVERSION-DCB-HEX-1
                                TITLE  CONVERSION DCB-HEX
7040                                ORG    $7040
                                DEBUT
7040 86    15    LDA    #$15
7042 87    7101  STA    $7101
7045 84    F0    ANDA   #$F0    (A)=X0=X*16
7047 44                   LSRA                   (A)/2 =X*8
7048 44                   LSRA                   (A)/2 =X*4
7049 87    7103  STA    $7103   TAMPON
704C 44                   LSRA                   (A)/2 =X*2
704D 88    7103  ADDA   $7103
7050 87    7103  STA    $7103
7053 86    7101  LDA    $7101
7056 80    7103  SUBA   $7103
7059 87    7101  STA    $7101
705C 3F                                SWI
                                7040    END    DEBUT
```

Le test de ce programme conduit aux résultats ci-dessous :

```
15 → 0F
27 → 1B    ( 27 = 16 + 11 ou B )
73 → 49    ( 73 = 16 * 4 + 9 )
99 → 63    ( 99 = 16 * 6 + 3 )
09 → 09
```

Une fois ce programme testé, **comment l'insérer** dans celui de la multiplication ?... c'est l'objet du chapitre suivant.

6

MULTIPLICATION DÉCIMALE DE 2 NOMBRES DE 2 CHIFFRES

BUT : *Synthèse des programmes précédents, sous-programmes.*

INSTRUCTIONS UTILISEES : *BSR, RTS, BEQ, ADC, BCC.*

ADRESSAGE : *Etendu.*

Le premier problème que nous avons à résoudre, c'est celui de l'insertion du programme de conversion dans le programme de la multiplication. Nous pouvons le faire en insertion « directe » c'est-à-dire que la conversion sera une partie du programme multiplication, les séquences d'instructions ne pouvant être dissociées.

C'est la méthode « linéaire » qui donne les programmes les plus rapides. Après :

```
STA    $7102
```

nous écrivons :

```
        ANDA    #$F0
        ,,
        ,,
        ,,
        ,,
        STA     $7101
        CLR     $7102
        ,,
        ,,
        SWI
```

La deuxième méthode consiste en la création d'un *sous-programme (procédure, subroutine)*. C'est cette technique (purement didacticielle ici) que nous employons.

Cette méthode est essentiellement utilisée pour :

- réduire l'espace mémoire nécessaire à un programme (on « *gagne* » des octets mais on « *perd* » du temps)
- clarifier un programme et permettre une mise au point morceau par morceau (technique *modulaire*).

Question : Qu'est-ce qu'un sous-programme ?

C'est un ... programme dont la structure permet l'insertion dans n'importe quel autre programme, au moment où ce dernier en a besoin. Comme un sous-programme peut être *appelé* à n'importe quel instant, les instructions de *branchement* sont à des adresses *inconnues* du programmeur ayant mis au point le sous-programme, il faut terminer un sous-programme par une instruction de *retour* (RTS). Pour que le retour s'effectue correctement il faut que l'adresse de retour,

c'est-à-dire l'adresse de l'instruction qui suit celle *d'appel*, soit mémorisée. Il n'existe pas de registre prévu à cet effet, car, un sous-programme peut appeler un sous-programme qui peut...

Aussi, dans le cas d'écriture de logiciel avec procédures devons-nous définir une zone mémoire de stockage temporaire des adresses de retour. Cette zone mémoire *vive* (en RAM) est appelée *pile* et est gérée grâce à un registre pointeur de pile ou *stack pointer* (S).

Lors de l'appel au sous-programme l'adresse de retour est stockée en pile (2 octets) et le contenu du registre S **décrémente** de 2. Lors du retour PC est chargé à l'adresse mémorisée et le contenu de S **incrémente** de 2.

Il faut donc **initialiser** S si l'on désire utiliser des sous-programmes.

L'emploi de procédures permet une écriture plus claire, une économie d'octets mais exige de la mémoire vive et ralentit l'exécution du programme.

Pour fixer les idées nous supposons que le moniteur n'initialise pas **notre** pointeur de pile afin de prendre de bonnes habitudes.

La première instruction du programme *principal* est donc :

LDS # \$8000

(\$8000 est une adresse située en *haut* de la RAM utilisateur).

L'instruction d'appel sera :

BSR CONVER

CONVER étant l'étiquette définissant le programme de CONVERsion, qui se termine par

RTS

venant remplacer SWI qui a servi à la mise au point.

Mais nous n'appellerons CONVER que si nécessaire, aussi après avoir chargé les 2 nombres en 7100 et 7101 (adresses en hexadécimal), nous utiliserons un *masque* pour savoir si la conversion DCB-HEX, est nécessaire. Le multiplicateur étant dans l'accumulateur A le masquage sera réalisé par l'exécution de l'instruction

ANDA # \$F0

Si le chiffre des dizaines est nul (nombre inférieur à 10) la conversion n'est pas nécessaire. Dans ce cas le contenu de l'accumulateur A sera nul car :

$$\begin{array}{r} 0000 \text{ yyyy} \\ \text{ET } 1111 \text{ 0000} \\ \hline = 0000 \text{ 0000} \end{array}$$

Le résultat étant nul le flag Z est mis à 1, aussi utiliserons-nous l'instruction BEQ (*Branchement SI Equal*) pour « sauter » l'instruction d'appel au sous-programme, BSR, car le 6809 ne possède pas d'instruction d'appel *conditionné* — ce n'est pas le seul microprocesseur dans ce cas. Nous devons donc écrire la séquence suivante :

ANDA # \$F0
BEQ BOUCLE
BSR CONVER

Nous profitons de cette modification de programme pour prévoir le cas où le multiplicateur est nul. Ceci est facilité par le fait que l'instruction de stockage (STr où r est un registre de 8 ou 16 bits) — comme celle de chargement LDr — affecte le flag Z. Donc après :

STA # \$7101

✱

écrivons-nous

```
BEQ  FIN
```

Mais cela n'est correct que si la case mémoire contenant le résultat est chargée à 0, nous devons donc mettre :

```
CLR  #$7102
```

avant le chargement de a et b.
Nous obtenons donc le listage suivant :

7000			ORG	\$7000	
7000	10CE	8000	DEBUT	LDS	##8000 INIT. PILE
7004	7F	7102		CLR	\$7102
7007	86	03		LDA	##3
7009	87	7100		STA	\$7100
700C	86	15		LDA	##15
700E	87	7101		STA	\$7101
7011	27	15		BEQ	FIN b=0
7013	84	F0		ANDR	##F0
7015	27	02		BEQ	BOUCLE
7017	8D	27		BSR	CONVER
7019	86	7102	BOUCLE	LDA	\$7102
701C	88	7100		ADDA	\$7100
701F	19			DAA	
7020	87	7102		STA	\$7102
7023	7A	7101		DEC	\$7101
7026	26	F1		BNE	BOUCLE
7028	3F		FIN	SWI	
*CONVERSION-DCB-HEX-1					
7040			ORG	\$7040	
7040	44		CONVER	LSRA	(A)/2 =X*8
7041	44			LSRA	(A)/2 =X*4
7042	87	7103		STA	\$7103 TAMPON
7045	44			LSRA	(A)/2 =X*2
7046	88	7103		ADDA	\$7103
7049	87	7103		STA	\$7103
704C	86	7101		LDA	\$7101
704F	80	7103		SUBR	\$7103
7052	87	7101		STA	\$7101
7055	39			RTS	
	7000		END	DEBUT	

L'exécution du programme en pas-à-pas jusqu'à BOUCLE, dans le cas où b est supérieur à 9 vous permettra d'observer le travail du 6809 lors de l'appel de CONVER.

La valeur initiale de (S) est 8000, elle passe à 7FFE soit $8000 - 2$.

L'adresse de retour après exécution de CONVER est celle de BOUCLE c'est-à-dire 7019. Elle est stockée en 7FFE et 7FFF.

Pour en être convaincu il suffit de lire le contenu des cases mémoire d'adresse (S) et (S) + 1, **après branchement** au sous-programme :

```
#7FFE/ 70
7FFF/ 19
```

Après exécution de l'instruction RTS le contenu de PC est 7019 et celui de S 8000. La pile « s'est vidée » — la **pile (stack)** est la zone mémoire **vive** « gérée » par S.

Il est nécessaire que vous ayez bien compris le rôle de S.

Mais ce programme ne « marche » pas pour des nombres de 2 chiffres — **essayez...**

Quelles sont les modifications à apporter ?

Il faut prévoir un résultat comptant 4 chiffres soit 2 octets stockés en 7102 pour l'octet des milliers et des centaines (octet de *poids fort*), et en 7103 pour l'octet des dizaines et des unités (octet de *poids faible*). Nous adoptons ici le principe du 6809 qui stocke un mot de 16 bits en mémoire avec l'octet de poids fort à l'adresse « basse » — ce n'est pas le cas de tous les microprocesseurs.

Nous écrivons donc :

```
CLR  $7102
CLR  $7103
```

Il faut évidemment traiter les *centaines*. **Quand** apparaissent-elles ?... après DAA, par exemple :

$$\begin{array}{rcl}
 & & \boxed{1} \\
 & & \swarrow \quad \searrow \\
 59 & = & 0101 \quad 1001 \\
 + 59 & = & 0101 \quad 1001 \\
 \hline
 B2 & = & 1011 \quad 0010 \\
 DAA \rightarrow + 66 & = & 0110 \quad 0110 \quad \text{avec } H = 1 \\
 \hline
 & & \boxed{1} \quad 0001 \quad 1000 = \boxed{1} \quad 18
 \end{array}$$

Remarquez que DAA ajoute, ici, 66 car :

- le flag H vaut 1
- le quartet de poids fort (B) dépasse 9.

La centaine qui apparaît est mémorisée dans le flag C (carry) et on ne peut avoir **qu'une** centaine à la fois. En effet le plus grand nombre décimal de 2 chiffres est 99, et :

$$99 + 99 = 1 \ 98$$

Nous avons 2 possibilités pour traiter les centaines :

- tester la valeur du carry
- utiliser une instruction qui tient compte de la valeur du carry.

Dans les 2 cas il faut ajouter 1, en décimal, au contenu de la case mémoire d'adresse 7102.

La première solution, qui vient immédiatement à l'esprit, consiste en un branchement conditionné sur C, par exemple :

BCC SUITE

qui branche à SUITE si le **carry** vaut 0 (*Carry Clear*) évitant la séquence d'instructions qui traite le cas où C vaut 1.

Nous aurions ainsi la séquence suivante :

```

      ”
      ”
      STA      $7103      rangement des dizaines et unités
      BCC      SUITE
      LDA      $7102      (A) = milliers et centaines
      ADDA     # $01
      DAA
      STA      $7102
SUITE DEC      $7101
      ”
      ”
```

Ici, il faut faire 3 remarques :

- 1 — DAA ne travaille que sur le contenu de l'accumulateur A, il faut donc charger les centaines dans A.
- 2 — Pour augmenter le contenu d'une case mémoire on peut l'**incrémenter** en utilisant l'instruction INC, mais celle-ci **n'affecte pas** le Carry, qui vaut 1... DAA ajouterait donc automatiquement 60.
- 3 — STA et LDA n'affectent pas le Carry.

Nous vous proposons la deuxième solution, moins évidente, mais qui présente l'intérêt de donner un exemple de ce qu'il faut faire pour obtenir des programmes courts et rapides.

Nous allons ajouter systématiquement le Carry aux centaines. Si celui-ci vaut 0 il n'y aura pas modification de leur nombre, par contre si C vaut 1, nous incrémentons en décimal le contenu de la case 7102.

Cette instruction est :

ADCA

qui ajoute au contenu de A celui de l'opérande indiqué **et** le **carry**. Pour résoudre notre problème, l'opérande sera ... 0, nous écrirons donc :

```

      ”
      STA      $7103
      LDA      $7102
      ADCA     # 0
      DDA
      STA      $7102
      DEC      $7101
      ”
```

Le listage est alors :

```

      *MULTIPLICATION DECIMALE DE 2 NOMBRES
      *      DE 2 CHIFFRES

      TITLE  MULTIPLICATION-3

7000      ORG      $7000

7000 10CE 8000      DEBUT  LDS      $$8000  INIT. PILE
7004 7F   7102      CLR      $7102
7007 7F   7103      CLR      $7103
700A 86   15        LDA      $$15
700C 87   7100      STA      $7100
700F 86   15        LDA      $$15
7011 87   7101      STA      $7101
7014 27   1E        BEQ      FIN          b=0
7016 84   F0        ANDA     $$F0
7018 27   02        BEQ      BOUCLE
701A 80   24        BSR      CONVER
701C 86   7103      BOUCLE  LDA      $7103  POIDS FAIBLE
701F 88   7100      ADDA     $7100
7022 19                      DAA
7023 87   7103      STA      $7103
7026 86   7102      LDA      $7102
7029 89   00        ADCA     $$0          CENT. + RETENUE
702B 19                      DAA
702C 87   7102      STA      $7102
702F 7A   7101      DEC      $7101
7032 26   E8        BNE      BOUCLE
7034 3F      FIN     SWI

      *CONVERSION-DCB-HEX-1

7040      ORG      $7040

7040 44      CONVER  LSRA          (A)/2 =X*8
7041 44      LSRA          (A)/2 =X*4
7042 87   7104      STA      $7104  TAMPON
7045 44      LSRA          (A)/2 =X*2
7046 88   7104      ADDA     $7104
7049 87   7104      STA      $7104
704C 86   7101      LDA      $7101
704F 80   7104      SUBA     $7104
7052 87   7101      STA      $7101
7055 39      RTS

      7000      END      DEBUT

```

7

MULTIPLICATION

INSTRUCTION UTILISEE : *TFR*.

ADRESSAGE : *Direct*.

Pour nous reposer un peu nous allons aborder un problème très simple : l'adressage direct.

L'adressage direct est obtenu en utilisant le registre DP (*Direct Page Register*) qui permet l'adressage de 256 cases mémoires (une page). Le contenu de DP est initialisé à la valeur de l'octet de poids *fort* des adresses souhaitées, l'octet de poids *faible* est donné dans les instructions... **Attention** aux erreurs de frappe : il ne faut donner qu'*un* octet !

Les instructions nécessaires sont :

```
LDA # $71  octet de poids fort  
TFR A,DP
```

TFR A,DP recopie (pas de destruction) le contenu de A dans DP. On peut utiliser TFR pour n'importe quelle paire de registres de 8 ou 16 bits.

La mise à 0 du contenu de la case d'adresse 7102 s'effectue maintenant ainsi :

```
CLR $02
```

Le listage devient :

```

#MULTIPLICATION DECIMALE DE 2 NOMBRES
*   DE 2 CHIFFRES
*   AVEC ADRESSAGE "DIRECT"

                TITLE  MULTIPLICATION-4

7000                ORG      $7000

7000 10CE 8000      DEBUT  LDS      $$8000  INIT. PILE
7004 86  71        LDA      $$71    INITIALISATION
7006 1F  98        TFR      A,DP     DE DP
7008 0F  02        CLR      $02     ADRESSAGE DIRECT
700A 0F  03        CLR      $03
700C 86  15        LDA      $$15
700E 97  00        STA      $00
7010 86  15        LDA      $$15
7012 97  01        STA      $01
7014 27  18        BEQ      FIN      b=0
7016 84  F0        ANDA     #$F0
7018 27  02        BEQ      BOUCLE
701A 80  24        BSR      CONVER
701C 96  03        BOUCLE  LDA      $03
701E 98  00        ADDA     $00
7020 19          DAA
7021 97  03        STA      $03
7023 96  02        LDA      $02
7025 89  00        ADCA     #$0      CENT. + RETENUE
7027 19          DAA
7028 97  02        STA      $02
702A 0A  01        DEC      $01
702C 26  EE        BNE      BOUCLE
702E 3F          FIN    SWI

#CONVERSION-DCB-HEX-1

7040                ORG      $7040

7040 44          CONVER  LSRA          (A)/2 =X*8
7041 44          LSRA          (A)/2 =X*4
7042 97  04        STA      $04      TAMPON
7044 44          LSRA          (A)/2 =X*2
7045 98  04        ADDA     $04
7047 97  04        STA      $04
7049 96  01        LDA      $01
704B 90  04        SUBA     $04
704D 97  01        STA      $01
704F 39          RTS

                7000      END      DEBUT

```

Nous avons *gagné* 12 octets par rapport au listage précédent (6 dans le programme principal et 6 dans le sous-programme).

8

ENTRÉE DE NOMBRES DE DEUX CHIFFRES ET AFFICHAGE

BUT : *Utilisation de sous-programmes moniteur.*

INSTRUCTIONS UTILISEES : *JSR, BRA, TST, SUB, CMP, OR.*

ADRESSAGE : *Symbolique.*

Problème : Entrer des nombres au clavier en les affichant.

Il est évident que le moniteur de votre ordinateur comporte un programme « Lecture d'une Touche » et un programme « Affichage d'un Caractère ». Ils ont été dénommés respectivement :

- GETC (*GET a Character*) qui commence en E806 pour TO7 et TO7-70
- PUTC (*PUT a Character*) qui commence en E803 pour TO7 et TO7-70

Pour utiliser un sous-programme moniteur, connaître « son adresse » n'est pas suffisant. Il faut également savoir les conditions d'*entrée* et celles de *sortie*.

GETC dans notre cas (chiffres décimaux) ne présente aucune condition d'accès. Si aucune touche n'est pressée, B est chargé à 0, sinon il contient le code ASCII (*American Standard Code for Information Interchange*).

Le code ASCII attribue un mot binaire de 7 bits (00 à 7F) à chaque caractère alphanumérique et à quelques commandes concernant la communication entre systèmes informatisés (voir tableau III).

PUTC émet vers l'écran (terminal) le contenu de B qui doit être un code ASCII. Compte tenu de ces informations, nous pouvons écrire un court programme d'essai qui consiste à attendre qu'une touche soit pressée pour afficher son caractère — attention à **ENTREE**, **RAZ**,...

L'accès aux sous-programmes moniteur utilise **OBLIGATOIREMENT** l'instruction :

JSR sous-programme

Nous aurons donc la séquence suivante :

```

BOUCLE  LDS    # $8000 init. S
        JSR    GETC
        TSTB
        BEQ    BOUCLE
        JSR    PUTC
        BRA    BOUCLE

```

On ne sort de ce programme que par un « RESET » ou INITIALISATION PROGRAMME !

L'instruction TSTB « positionne » les flags N (signe) et Z en fonction du contenu de B, et C est mis à 0, donc si aucune touche n'est pressée, (B) est nul, et BEQ nous branche à BOUCLE.

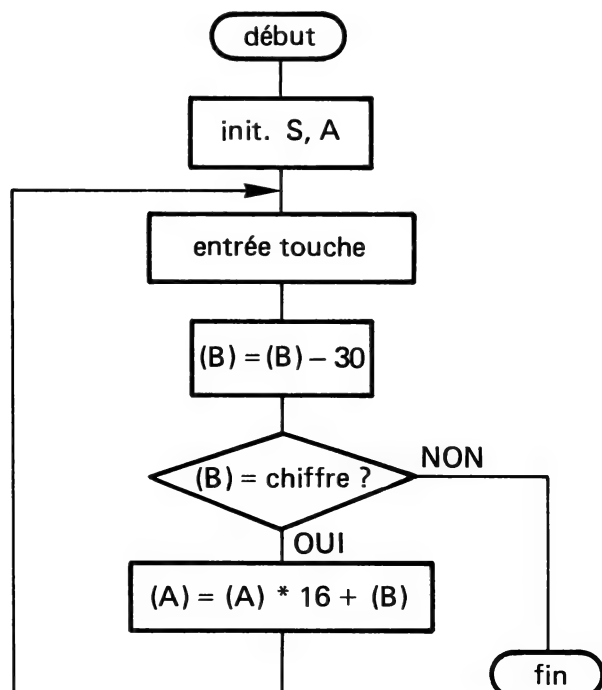
L'instruction BRA (*BRanch Always*) nous renvoie **toujours** à BOUCLE.

Essayez ce programme pour voir si en appuyant sur A vous avez un A sur votre écran... Pour « voir » les codes ASCII, il faut lire (B) à l'aide d'un point d'arrêt à l'instruction JSR PUTC.

Pour résoudre notre problème, il faut retirer 30 au code ASCII — les chiffres sont codés de 30 (0) à 39 (9) — et associer les chiffres pour en faire un nombre. C'est-à-dire qu'à chaque entrée d'un nouveau chiffre, il faut multiplier le nombre précédent par 16 (un 0 à droite) et ajouter le chiffre frappé.

Comment indiquer la fin du programme ? Le plus simple est de considérer qu'une touche qui ne correspond pas à un chiffre indique la fin du nombre. Il faut donc tester le contenu de B par rapport à 0 et 9, ce qui est résolu par l'instruction CMPB qui affecte les flags en conséquence **sans** détruire (B). Les instructions qui suivent CMP sont du type BMI (Branchement si négatif) ou BHI (Branchement si supérieur à — arithmétique non signée).

L'organigramme est le suivant :



Nous obtenons le listage suivant pour lequel nous avons adopté l'écriture *symbolique* grâce aux *directives* de l'assembleur.

- EQU permet de donner un nom à une donnée ou à une adresse
- FCB permet de fixer le contenu d'une case mémoire, ici 0 pour réserver une case TAMPON à la fin du programme qui est en RAM, car ADDA B n'existe pas.

Remarque : il faut initialiser A à 0, et BMI est mis après SUBB #\$30 car... 2F-30 est négatif.

```

* ENTREE DE NOMBRES AU CLAVIER
*LE REGISTRE A CONTIENT LE NOMBRE
* LIMITE A 2 CHIFFRES

      E806      GETC      EQU      $E806
      E803      PUTC      EQU      $E803

7000                                ORG      $7000

7000 10CE 0000      DEBUT  LDS      $$0000
7004 4F                                CLRA
7005 80      E806      ENTREE JSR      GETC
7008 5D                                TSTB
7009 27      FA                                BEQ      ENTREE
700B 80      E803                                JSR      PUTC
700E C0      30                                SUBB     #$30
7010 2B      10                                BMI      FIN
7012 C1      09                                CMPB     #$9
7014 22      0C                                BHI      FIN
7016 48                                ASLA
7017 48                                ASLA
7018 48                                ASLA
7019 48                                ASLA
701A F7      7023                                STB      TAMPON
701D B8      7023                                ADDA     TAMPON
7020 20      E3                                BRA      ENTREE
7022 3F                                FIN      SWI

7023 00                                TAMPON  FCB      0

      7000                                END      DEBUT
```


9

MESSAGE

BUT : *Utiliser le programme PUTC et l'.*

ADRESSAGE : *Indéxé Auto-incrémenté.*

Pour afficher un caractère quelconque, il faut mettre son code ASCII dans B et appeler PUTC. Il est donc très simple d'émettre un message. Il suffit d'écrire autant de fois qu'il le faut la séquence suivante :

```
LDB #$xx  xx est le code du caractère
JSR PUTC
```

soit 5 octets par caractère, sans compter le « compteur » de caractère...
Mais si l'on peut stocker les caractères déjà codés en mémoire et les charger un par un dans B, on gagne beaucoup de place.

Pour ce faire, nous avons l'adressage *indexé*, via X ou Y, qui peut être **post**-incrémenté ou **pré**-décrémenté. Dans le premier cas, le contenu du registre (X ou Y) est incrémenté **après** exécution de l'instruction ; dans le deuxième cas, la décrémentation a lieu **avant** l'exécution de l'instruction.

Ainsi :

```
LDB ,X+
```

met dans B le contenu de la case dont l'adresse est le contenu de X, **et** incrémente ce dernier.

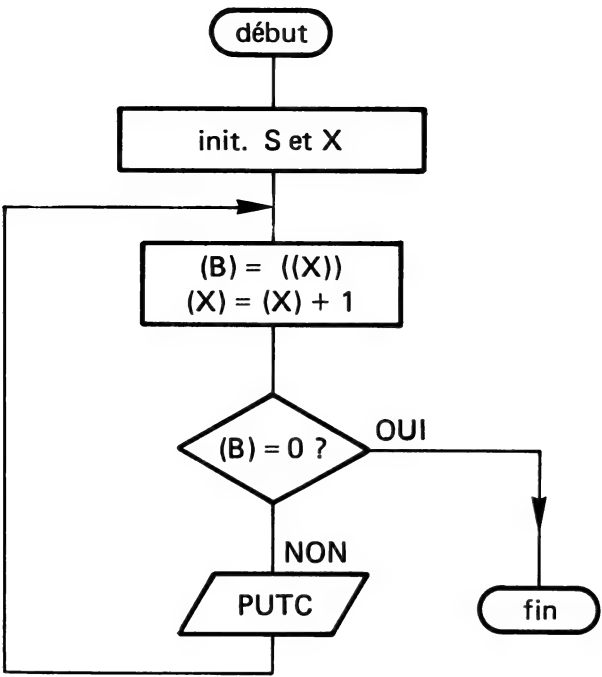
Ce qui, dans notre cas, permet de *pointer* la case suivante.

Pour indiquer la fin du message à émettre, on peut :

- compter les caractères
- mettre un caractère de fin.

C'est la deuxième solution que nous adopterons car c'est la plus simple et la plus souple, le programme pourra devenir un sous-programme sans modification importante. Le caractère de FIN sera `00`, caractère qui ne provoque rien sur l'écran (`00` = NUL) et en raison de la modification des flags par l'instruction LDB, facilement repérable.

L'organigramme est :



Le listage est :

```
*      MESSAGE
      E803  PUTC  EQU  $E803
7000
      ORG    $7000
7000 10CE 8000  DEBUT  LDS  #$8000
7004 8E   700F    LDX  #MESSAGE
7007 E6   80     BOUCLE LDB  ,X+
7009 B0   E803    JSR  PUTC
700C 26   F9     BNE  BOUCLE
700E 3F
      SWI

700F 4A 45 20 53  MESSAGE FCC  'JE SUIS LA PLUS BELLE'
7013 55 49 53 20
7017 4C 41 20 50
701B 4C 55 53 20
701F 42 45 4C 4C
7023 45
7024 0A 0D 00      FCB  $0A,$0D,0

      *Nous n'avons droit qu'à SIX lettres
      *Pour les ETIQUETTES

7000      END  DEBUT
```

dans lequel apparaît la *directive* FCC qui permet d'entrer des codes ASCII. Le message se termine par 0A et 0D qui sont des commandes amenant le curseur en début de ligne (0D = Retour Chariot) suivante (0A = saut de ligne ou *Line Feed*).

10

AFFICHAGE D'UN NOMBRE DE QUATRE CHIFFRES

BUT : Utiliser le programme précédent, préparer le suivant.

INSTRUCTION UTILISEE : *MUL*.

ADRESSAGE : *Etendu, Indexé et Symbolique.*

Problème : Afficher un nombre de 4 chiffres (2 octets) stocké en 7102 et 7103. L'octet de poids fort est en 7102.

Il faut séparer les chiffres et leur ajouter 30 pour obtenir leur code ASCII. Ainsi, en supposant que le nombre à afficher est 4567, il faut obtenir successivement 34, 35, 36 et 37.

Dans de tels cas, le plus simple est de construire une « table » des données à « sortir », que le récepteur soit un écran, une imprimante, une disquette... Le récepteur recevra les caractères, sans les temps morts dus aux calculs, et nous pourrons utiliser le programme précédent.

Le premier travail à exécuter est donc la construction de la table.
Nous utiliserons le registre D, de 16 bits, qui est la réunion de A (poids fort) et de B, et le fait que l'instruction

MUL

à pour fonction la multiplication de (A) par (B), le résultat étant rangé dans... D. La multiplication est, évidemment, **binaire et non signée** :

$$(A) * (B) = (D)$$

Cette propriété va nous permettre de séparer simplement un octet en 2 quartets. Si A contient 45 et B 10, le produit de (A) par (B) sera :

04 50

avec 04 dans A et 50 dans B.

Pour obtenir le code ASCII, il suffit d'ajouter 30 au contenu de A, quant à celui de B, il faudra préalablement commander 4 permutations circulaires sur 8 bits.

Le stockage des résultats, la mise en table, est obtenu en utilisant l'adressage indexé auto-incrémenté de manière à pointer la case suivante après chaque rangement. Ce qui donne le sous-programme STOCK.

La première partie du programme charge le nombre à afficher. Après constitution de la table des octets à afficher, nous complétons cette dernière par un saut de ligne et un « retour chariot », peu importe l'ordre, obtenus à l'aide des codes ASCII 0A et 0D. Cette opération est réalisée par le chargement *immédiat* de D et son stockage via (X). Remarquez qu'il faut ici une **double** incrémentation de (X) si l'on veut pointer la case suivante afin de ranger le mot « FIN » qui est pour nous (voir le programme précédent) : 00.

Nous obtenons donc le listage suivant :

*AFFICHAGE D'UN NOMBRE DE 4 CHIFFRES					
*RANGES EN 7102 ET 7103					
7000			ORG	\$7000	
	E803	PUTC	EQU	\$E803	
7000	10CE	8000	DEBUT	LDS	##8000
7004	86	45		LDA	##45
7006	87	7102		STA	\$7102
7009	86	67		LDA	##67
700B	87	7103		STA	\$7103
700E	8E	7043		LDX	#TAMPON
7011	86	7102		LDA	\$7102
7014	8D	7028		JSR	STOCK
7017	86	7103		LDA	\$7103
701A	8D	7028		JSR	STOCK
701D	0C	000A		LDD	##000A
7020	ED	81		STD	,X++
7022	6F	84		CLR	,X
7024	8D	7038		JSR	AFFICH
7027	3F			SWI	
7028	06	10	STOCK	LDB	##10
702A	3D			MUL	
702B	8B	30		ADDA	##30
702D	A7	80		STA	,X+
702F	54			LSRB	
7030	54			LSRB	
7031	54			LSRB	
7032	54			LSRB	
7033	0B	30		ADDB	##30
7035	E7	80		STB	,X+
7037	39			RTS	
7038	8E	7043	AFFICH	LDX	#TAMPON
703B	E6	80	AFF0	LDB	,X+
703D	8D	E803		JSR	PUTC
7040	26	F9		BNE	AFF0
7042	39			RTS	
7043	00		TAMPON	FCB	0
	7000		END	DEBUT	

Le chargement de 7102 et 7103 peut être obtenu en remplaçant les deux LDA par des appels au programme 8.

11

MULTIPLICATION DÉCIMALE DE DEUX NOMBRES DE DEUX CHIFFRES ENTRÉS AU CLAVIER, AFFICHAGE DU RÉSULTAT

BUT : *Synthèse des programmes précédents.*

La réunion des programmes 7 (multiplication), 8 (entrée au clavier) et 10 (affichage) donne le listage suivant :

```

*MULTIPLICATION DECIMALE DE 2 NOMBRES
*      DE 2 CHIFFRES
*      AVEC ADRESSAGE "DIRECT"
*      ENTREE CLAVIER ET AFFICHAGE

7000                                ORG      $7000

                                E803      PUTC      EQU      $E803
                                E806      GETC      EQU      $E806

7000 10CE 8000      DEBUT  LDS      $$8000  INIT. PILE
7004 86      71      LDA      #$71      INITIALISATION
7006 1F      88      TFR      A,DP      DE DP
7008 0F      02      CLR      $02      ADRESSAGE DIRECT
700A 0F      03      CLR      $03
700C 8D      7035     JSR      ENTREE
700F 97      00      STA      $00
7011 8D      7035     JSR      ENTREE
7014 97      01      STA      $01
7016 27      18      BEQ      FINI      b=0
7018 84      F0      ANDA     #$F0
701A 27      02      BEQ      BOUCLE
701C 8D      36      BSR      CONVER
701E 96      03      BOUCLE LDA     $03
7020 96      00      ADDA     $00
7022 19      DAA
7023 97      03      STA      $03
7025 96      02      LDA      $02
7027 89      00      ADCA     #$0      CENT. + RETENUE
7029 19      DAA
702A 97      02      STA      $02
702C 0A      01      DEC      $01
702E 26      EE      BNE      BOUCLE
7030 8D      7068     FINI  JSR      AFFICH
7032 20      0B      BRA      DEBUT

```

7035	4F		ENTREE	CLRA		
7036	8D	E806	ENTRE	JSR	GETC	
7039	5D			TSTB		
703A	27	FA		BEQ	ENTRE	
703C	8D	E803		JSR	PUTC	
703F	00	30		SUBB	#\$30	
7041	28	10		BMI	FIN	
7043	C1	09		CMPS	#\$9	
7045	22	0C		BHI	FIN	
7047	48			ASLA		
7048	48			ASLA		
7049	48			ASLA		
704A	48			ASLA		
704B	F7	709D		STB	TAMPON	
704E	86	709D		ADDA	TAMPON	
7051	20	E3		BRH	ENTRE	
7053	39		FIN	RTS		
7054	44		CONVER	LSRA		(A)/2 =X*8
7055	44			LSRA		(A)/2 =X*4
7056	87	709D		STA	TAMPON	
7059	44			LSRA		(A)/2 =X*2
705A	86	709D		ADDA	TAMPON	
705D	87	709D		STA	TAMPON	
7060	96	01		LDA	\$01	
7062	80	709D		SUBA	TAMPON	
7065	97	01		STA	\$01	
7067	39			RTS		
7068	8E	709D	AFFICH	LDX	#TAMPON	
706B	86	7102		LDA	\$7102	
706E	8D	7082		JSR	STOCK	
7071	86	7103		LDA	\$7103	
7074	8D	7082		JSR	STOCK	
7077	0C	0D0A		LDD	#\$0D0A	0D=RC, 0A=LF
707A	E0	01		STD	,X++	2 OCTETS A RANGER
707C	6F	04		CLR	,X	+ INUTILE
707E	8D	7092		JSR	AFF	
7081	39			RTS		
7082	06	10	STOCK	LDB	#\$10	
7084	3D			MUL		
7085	8B	30		ADDA	#\$30	CODE ASCII
7087	A7	80		STA	,X+	
7089	54			LSRB		
708A	54			LSRB		
708B	54			LSRB		
708C	54			LSRB		
708D	0B	30		ADDB	#\$30	
708F	E7	80		STB	,X+	
7091	39			RTS		
7092	8E	709D	AFF	LDX	#TAMPON	
7095	E6	80	AFF0	LDB	,X+	
7097	8D	E803		JSR	PUTC	
709A	26	F9		BNE	AFF0	
709C	39			RTS		
709D	00		TAMPON	FCB	0	
		7000		END	DEBUT	

La fin (SWI) des programmes 8 et 10 est remplacée par un retour RTS, et les divers LDA nécessaires à la mise au point deviennent JSR ENTREE.

Les nombres seront séparés par *, le deuxième terminé par =, et le résultat apparaît immédiatement. Le programme ne prend en compte que les 2 chiffres précédant * et =. Après affichage d'un résultat, le programme attend deux nouveaux nombres.

Nous aurons, par exemple :

$$15 * 15 = 0225$$

$$23415 * 789415 = 0225$$

Attention le curseur n'apparaît pas...

12

MULTIPLICATION HÉXADECIMALE SIGNÉE

BUT : Apprendre à calculer avec des nombres réels en binaire.

INSTRUCTIONS UTILISEES : COM, NEG, BPL, BCS.

ADRESSAGE : Direct.

Problème : Multiplier 2 nombres, binaires, signés de 8 bits.

Il est évident que le problème n'existe ici que par la présence du mot **signe**. En effet, nous disposons d'une instruction MUL, que nous avons déjà utilisée, mais elle ne permet que la multiplication, binaire, **non** signée.

C'est-à-dire que nous aurons, avec MUL, les résultats suivants :

$$\begin{aligned}FF * FF &= FE01 \\ 80 * 40 &= 2000\end{aligned}$$

c'est-à-dire :

$$\begin{aligned}255 * 255 &= 65025 \\ 128 * 64 &= 8192\end{aligned}$$

Tous les nombres sont considérés **positifs**.

Par contre, en arithmétique signée, un nombre est dit **négalif** dès lors que son bit de signe vaut 1 et il est écrit sous sa forme **complément à 2**.

Dans ce cas, le plus grand nombre positif est :

$$0111\ 1111 \text{ soit } 7F \text{ ou } +127$$

le plus petit nombre positif est évidemment 0.

Quant aux nombres **négalifs** ils « vont » de :

$$1000\ 0000 \text{ ou } 80 \text{ à } 1111\ 1111 \text{ ou } FF$$

Que représentent 80 et FF ?

Pour le savoir il faut passer au **complément à 2** qui consiste à ajouter 1 au **complément à 1**, obtenu en remplaçant les 1 par des 0 et vice versa — si N est un nombre, on note \bar{N}_1 le complément à 1 et \bar{N}_2 le complément à 2.

Ainsi, FF représente :

$$\begin{aligned}N &= 1111\ 1111 = FF \\ \bar{N}_1 &= 0000\ 0000 = 00 \text{ on ajoute 1 pour avoir} \\ \bar{N}_2 &= 0000\ 0001 = 01\end{aligned}$$

dont FF est équivalent à « -1 » en arithmétique signée.

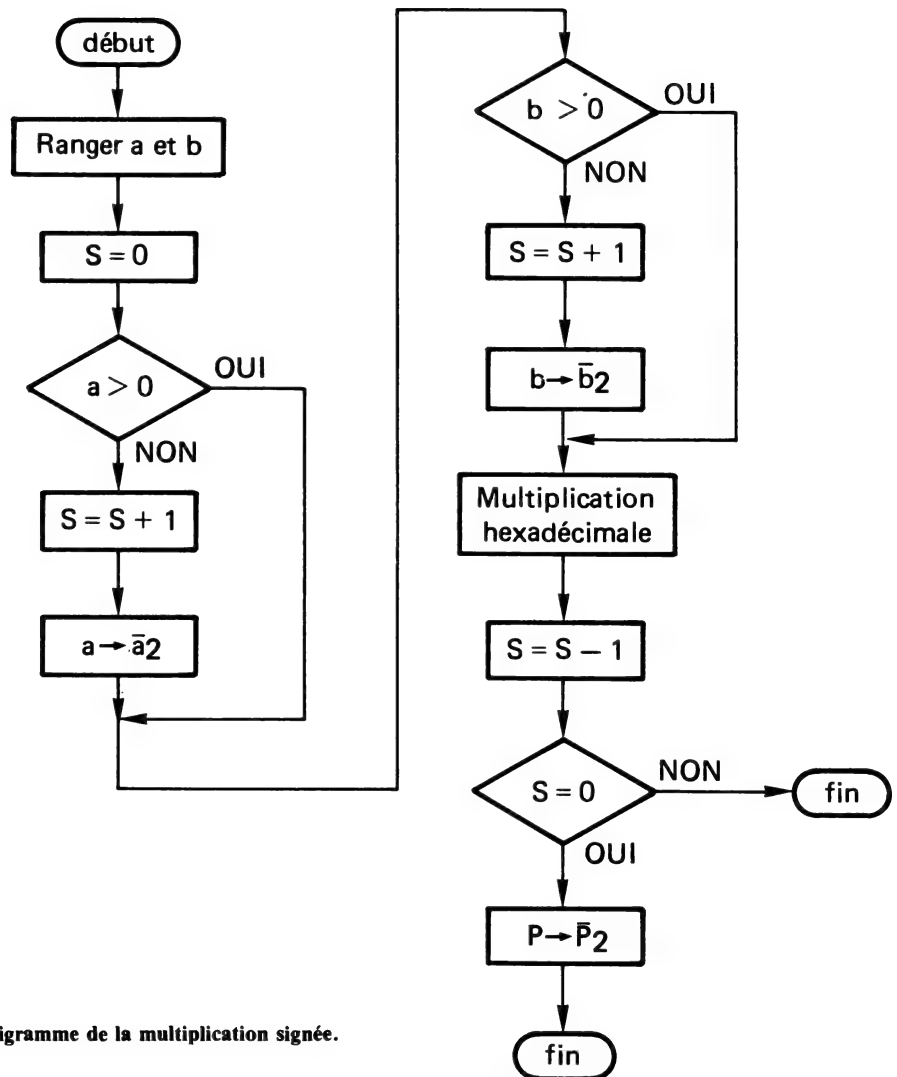
De même on trouve pour 80 :

$$\begin{aligned}N &= 1000\ 0000 = 80 \\ \bar{N}_1 &= 0111\ 1111 = 7F \\ \bar{N}_2 &= 1000\ 0000 = 80\end{aligned}$$

donc 80 est équivalent à « -128 ».

En arithmétique signée, nous pouvons écrire, sur 8 bits, les nombres de +127 à -128 (256 possibilités). En arithmétique **non** signée, nous comptons de 0 à 255. Pour réaliser une multiplication **signée**, nous allons utiliser le principe du *sémaphore* (indicateur) qui indiquera le signe du résultat. Ce sémaphore, initialisé à 0, est incrémenté à chaque fois qu'un nombre est négatif (bit de signe à 1). Le nombre est complémenté à 2. La multiplication **non** signée est exécutée, puis on lit l'état du sémaphore, s'il est à 1 le résultat est négatif et doit être complémenté à 2, sinon (0 ou 2) le résultat est positif.

L'organigramme est donc le suivant :



Organigramme de la multiplication signée.

Le test de signe sera effectué à l'aide de l'instruction :

BPL Branchement SI PLus

La complémentation à 2 d'un nombre de 8 bits est obtenue par l'instruction :

NEG NEGation

Le test du sémaphore est réalisé par décrémentation de ce dernier, si le résultat est **nul** cela signifie que le produit est **négatif**. Il faut alors complémenter à 2 le contenu de D.

La complémentation à 2 d'un nombre de 16 bits s'opère (microprocesseur 8 bits) en 3 temps :

- complémentation à 1 de l'octet de poids fort
- complémentation à 2 de l'octet de poids faible
- incrémentation de l'octet de poids fort si l'opération précédente l'exige.

Par exemple :

1) Si le nombre est : 1600H

$$1600 = 0001\ 0110\ 0000\ 0000$$

complément à 1 de 16 :

$$1110\ 1001$$

complément à 1 de 00 :

$$1111\ 1111$$

complément à 2 de 00 (FF + 1) :

$$1\ 0000\ 0000$$

Il faut incrémenter l'octet de poids fort :

$$1110\ 1010$$

Le résultat est EA00 = -1600H

2) Le résultat est 15BC

le complément à 1 de 15 est :

$$1110\ 1010$$

le complément à 1 de BC est :

$$0100\ 0011$$

le complément à 2 de BC est donc :

$$0100\ 0100$$

sans dépassement. Le complément à 2 de 15BC est donc :

$$EA44 = -15BC$$

Le complément à 1 est obtenu par l'instruction :

COM

Le problème qui apparaît est : quelle instruction mettre après NEG qui complémente à 2 l'octet de poids faible ?

Contrairement aux exemples ci-dessus, il faut utiliser :

BCS Branchement SI Carry Set (égal à 1)

Pour le cas où il n'est **pas** nécessaire d'incrémenter l'octet de poids fort.

Pour simplifier le listage qui suit, le chargement des opérandes s'effectue en cours de travail, directement, à l'aide de LDA et LDB. Vous pouvez essayer de le faire en faisant appel au programme 8.

Il faudra alors charger une case mémoire car A et B sont utilisés dans le programme 8, puis recharger A.
N'oubliez pas l'initialisation de S.
Vous pourrez également faire appel au programme d'affichage 10.
Nous donnons ci-après 4 exemples concernant les 4 cas possibles.

Exemple 1 :
a = 52D = 0011 0100 = 34H
b = 107D = 0110 1011 = 6BH
P = 5564D = 0001 0101 1011 1100 = 15BCH

Exemple 2 :
a = -52D = 1100 1100 = CCH
b = 107D = 0110 1011 = 6BH
P = -5564D = 1110 1010 0100 0100 = EA44H

La multiplication, non signée (MUL seule), de CCH par 6BH, soit 204D par 107D, donne P = 5544H dont le bit de signe est 0 !

Exemple 3 :
a = -52D = 1100 1100 CCH
b = -107D = 1001 0101 95H
P = +5564D = 0001 0101 1011 1100 = 15BCH

Exemple 4 :
a = 44D = 0010 1100 2CH
b = -128D = 1000 0000 80CH
P = -5632D = 1110 1010 0000 0000 = EA00H

Pour ce cas, voir la complémentation à 2 d'une donnée de 16 bits, exposée plus haut.

```
#MULTIPLICATION HEXADÉCIMALE 8 BITS
#PAR 8 BITS SIGNÉE
#ENTRÉE DANS A ET B
#RESULTAT EN 7101 (POIDS FORT) ET
#7102

7000                                ORG    $7000

7000 86 71      DEBUT  LDA    #$71
7002 1F 8B      TFR    A,DP
7004 0F 00      CLR    $0      SEMAPHORE
7006 86 6B      LDA    #$6B
7008 2A 03      BPL    MUL0
700A 40          NEGB
700B 0C 00      INC    $0
700D 06 34      MUL0  LDB    #$34
700F 2A 03      BPL    MUL1
7011 50          NEGB
7012 0C 00      INC    $0
7014 3D      MUL1  MUL
7015 DD 01      STD    $01      RANGE RESULT.
7017 0A 00      DEC    $0
7019 26 08      BNE    FIN
701B 03 01      COM    $01
701D 00 02      NEG    $02
701F 25 02      BCS    FIN
7021 0C 01      INC    $01
7023 3F      FIN    SWI

7000                                END    DEBUT
```

13

SOMME DECIMALE DE N NOMBRES DE 2 OCTETS

BUT : *Utilisation de l'adressage indexé.*

Problème : Soient N nombres stockés en mémoire, comment obtenir leur somme décimale.

Si les nombres comptent 2 octets, c'est-à-dire 4 chiffres, il est probable que le résultat en comptera au moins 6 soit 3 octets. Il faut donc prévoir de comptabiliser les « dizaines de mille ».

Le principe de l'opération est simple, on additionne 2 nombres en commençant par les octets de poids faible, puis on ajoute le nombre suivant au résultat obtenu. On utilise donc :

ADD
pour les octets de poids faible, et

ADC
pour les octets de poids fort, pour tenir compte de la retenue (Carry) éventuelle de la première opération.

Le problème à résoudre est : comment lire les octets un par un et dans le *bon* ordre. La solution dépend de la méthode de stockage des nombres. Nous examinerons donc les 2 cas :

- octet de poids faible à l'adresse basse
- octet de poids faible à l'adresse haute.

1. Octet de poids faible à l'adresse basse

Si 1534 est stocké de cette façon en 7100, nous avons 34 en 7100 (adresse basse) et 15 en 7101.

Nous observons que, dans ce cas, l'octet de poids faible d'un nombre suit *immédiatement* l'octet de poids fort du nombre précédent :

poids faible 1	adresse X
poids fort 1	adresse X + 1
poids faible 2	adresse X + 2
poids fort 2	adresse X + 3
poids faible 3	adresse X + 4
poids fort 3	adresse X + 5

La lecture des octets peut donc être continue. De ce fait, l'adressage indexé, post-incrémenté, est tout indiqué.

Après avoir lu l'octet de poids faible N, on pointe l'octet de poids fort N, qu'on lit, pointant ainsi l'octet de poids faible N + 1.

Le programme est donc très simple. Nous rangeons le résultant dans le même ordre, et nous utilisons ADCA 0 pour incrémenter les dizaines de mille (voir programme 6).

Le listage est le suivant :

*SOMME DECIMALE DE N NOMBRES EN TABLE									
* ADRESSAGE INDEXE									
*NOMBRES RANGES EN 7100,7102,7104,7106									
*7108 RESULTAT EN 7120,7121,7122									
*OCTETS DE POIDS FAIBLE AUX ADRESSES									
*BASSES.									
7000				ORG	\$7000				
7000	8E	7100	DEBUT	LDX	##7100	ADR. 1ER NBRE			
7003	06	05		LOB	##5	NBRE DE NBRE			
7005	7F	7120		CLR	\$7120				
7008	7F	7121		CLR	\$7121				
700B	7F	7122		CLR	\$7122				
700E	86	7120	SUITE	LDA	\$7120				
7011	AB	80		ADCA	,X+	INDEXE			
7013	19			DAA					
7014	B7	7120		STA	\$7120				
7017	B6	7121		LDA	\$7121				
701A	A9	80		ADCA	,X+	OCTET SUIVANT			
701C	19			DAA					
701D	B7	7121		STA	\$7121				
7020	B6	7122		LDA	\$7122				
7023	89	00		ADCA	##0				
7025	19			DAA					
7026	B7	7122		STA	\$7122				
7029	5A			DECB					
702A	26	E2		BNE	SUITE				
702C	3F			SWI					
*NE PAS OUBLIER de charger les nombres!									
7000			END	DEBUT					

2. Octet de poids faible à l'adresse haute

C'est le principe de stockage, des mots de 16 bits, du 6809.

Si 1534 est stocké en 7100, nous avons 15 en 7100 et 34 en 7101. Si l'octet de poids fort du nombre N est à l'adresse X, l'octet de poids faible du nombre N+1 est à l'adresse X+3 !

poids fort 1	adresse X
poids faible 1	adresse X+1
poids fort 2	adresse X+2
poids faible 2	adresse X+3
poids fort 3	adresse X+4
poids faible 3	adresse X+5

Dans ce cas de figure, l'adressage indexé **post**-incrémenté n'est pas utilisable, il faut prendre l'adressage indexé **pré**-décrémenté, qui décrémenté le contenu du registre X ou Y **avant** d'effectuer l'instruction demandée. Ainsi :

LDA , - X

14

CHARGEMENT D'UN MESSAGE

BUT : *Création d'une « table » de caractères.*
INSTRUCTION UTILISEE : *LEA.*

Problème : Charger un message à partir du clavier.

Le programme GETC met dans B le code ASCII de la touche pressée, que nous stockerons en mémoire à l'aide de l'adressage indexé post-incrémenté. Si nous désirons compter les caractères du message, nous ne pouvons utiliser que A. Mais (A) peut être détruit par GETC et cela limite le nombre de lettres (ou signes de ponctuation) à 256. Pour dépasser cette limite, il faut utiliser un registre de 16 bits, mais nous ne disposons pas d'instruction d'incrémentation de 16 bits. Pour réaliser cette opération, nous utiliserons l'instruction :

LEA (*Load Effective Address*)

qui met dans le registre de 16 bits indiqué la *valeur* de l'*adresse* physique. C'est-à-dire que :

LEAY a,Y

mettra dans Y le contenu de Y augmenté de a, a pouvant être un nombre, positif ou négatif, ou le contenu d'un registre. Si a vaut 1 nous **incrémentons** (Y). La fin du message sera indiquée par un « Retour Chariot » — CR — code 0D et obtenue avec la touche **ENTREE**.

Le listing obtenu est le suivant :

```
*  ENTREE DE LETTRES AU CLAVIER
*  STOCKAGE EN MEMOIRE
*  LIMITES PAR "ENTREE"

      E806   GETC   EQU    $E806
      E803   PUTC   EQU    $E803

6800 10CE 8000   DEBUT   LDS    ##8000
6804 8E   681F   LDH     #LISTE
6807 108E 0000   LDY     #$0    COMPTEUR LETTRE
6808 8D   E806   CHARGE  JSR    GETC
680E 5D               TSTB
680F 27   FA     BEQ     CHARGE
6811 8D   E803   JSR     PUTC
6814 31   21     LEAY    1,Y
6816 E7   80     STB     ,X+
6818 C1   0D     CMPE    #$0D
681A 27   02     BEQ     SORTIE
681C 2D   ED     BRA     CHARGE
681E 3F               SORTIE SWI

681F 00               LISTE   FCB    0
      6800               END    DEBUT
```

15

CHARGEMENT D'UNE TABLE DE NOMBRES DE 16 BITS

BUT : Création d'une table de nombres, utilisation des piles.

INSTRUCTIONS UTILISEES : *ALS, PSH, PUL.*

Problème : Charger une table de nombres de **deux** octets.

Pour des raisons de souplesse, on ne peut utiliser le principe précédent pour des nombres. En effet, dans le programme précédent, **tout** est enregistré, donc le texte sera reproduit tel qu'il a été entré, avec ses fautes et ses **corrections**.

Ici, il faut construire un nombre de 4 chiffres et le stocker. La fin d'un nombre est signifiée par un caractère quelconque différent des chiffres décimaux. La fin de la liste, par la touche **ENTREE** codée 0D.

Pour construire le nombre, il faut disposer d'un registre de 16 bits que nous pourrions facilement multiplier par 16 (mettre un 0 à droite) pour insérer le chiffre tapé (voir programme 8).

Ce registre, compte tenu des instructions du 6809, ne peut être que D (A:B) en raison des opérations concernant A et B. Mais B est utilisé par le programme GETC, il faut donc *sauver* (B).

Pour cela, nous disposons d'une **pile** gérée par S (créée par l'initialisation de S). Mais avec le 6809, nous pouvons créer une **deuxième** pile gérée par U (Utilisateur). Cette pile, comme son nom l'indique, est à la disposition du programmeur, mais le processeur utilise celle gérée par S lors d'appels de sous-programmes et lors des retours.

En général, on utilisera la pile « U » pour les paramètres (données) à transmettre entre programmes et nous laisserons la pile « S » au processeur afin d'éviter quelques erreurs dues à une mauvaise gestion de nos piles.

La sauvegarde des données est réalisée grâce à l'instruction :

PSHU (ou PSHS) *Push*

Le contenu de U (ou S) est décrémenté de 1 ou 2 suivant la taille de la donnée à stocker en pile. A la restauration de la donnée réalisée par l’instruction :

PULU (ou PULS) Pull

l’opération inverse a lieu, c’est-à-dire que (U) ou (S) est incrémenté de 1 ou 2, et la donnée est chargée dans le registre indiqué. S’il y a eu plusieurs registres sauvegardés, par exemple :

PSHS A
,,
PSHS Y
,,
PSHS X
,,
PSHS B

Nous aurons dans la pile S :

(S ₀) – 6	(B)
	(X) haut
	(X) bas
	(Y) haut
	(Y) bas
(S ₀) – 1	(A)
(S ₀)

si l’on veut restaurer les registres, il faudra écrire :

PULS B
PULS X
PULS Y
PULS A

et non, bien qu’elle existe :

PULS B,X,Y,A

La suite de PULS restaure les registres dans l’ordre où ils ont été stockés en pile (opérations successives) ; l’instruction PULS B,X,Y,A restaure **toujours** les registres dans l’ordre : A,B,X,Y. (CC,A,B,DP,X,Y,U et PC). La sauvegarde, en une seule instruction, a évidemment lieu dans l’ordre inverse.

La pile est dite LIFO (*Last In First Out* — Dernier entré Premier sorti). Il faut donc bien prendre garde aux ordres de rangement et de restauration. Puisque la pile « monte » à chaque sauvegarde, il faut le même nombre de PULS (ou PULU) que de PSHS (ou PSHU) et **éviter** un PSH ou PUL dans une boucle... !

Notez que l’on peut écrire PSHS U et PSHU S.

L’existence de 2 piles de sauvegarde permet la séparation des variables. L’appel d’un sous-programme utilise automatiquement S : la pile se remplit à chaque appel et se vide à chaque retour. Si l’on ne dispose que d’une pile — c’est le cas le plus fréquent — il faut parfaitement la gérer pour éviter un retour à une adresse qui est une donnée, et vice versa (il existe de telles opérations volontaires).

Le 6809 possédant **deux** piles, on confie à la pile S la « gestion » du programme et à la pile U la gestion des données.

En raison de la composition de D (A,B) nous sommes, ici, obligés d’utiliser les 2 piles dans le sous-programme qui construit le nombre. Dès le début de ce programme, on sauve (D) puisque GETC et PUTC utilisent B. La construction du

nombre n'a lieu *que* si (B) est un chiffre décimal, on ne peut donc multiplier (D) par 16 qu'après les tests nécessaires ; ce qui explique le PSHS B avant PULU D.

Pour ajouter (B) à (D), il faut passer par l'adressage indexé. Nous savons que (B) est à l'adresse (S), donc on peut écrire :

```
ADDB ,S
```

mais il faut préparer le **retour**, donc incrémenter (S) de 1. Nous employons donc l'adressage indexé post-incrémenté :

```
ADDB ,S+
```

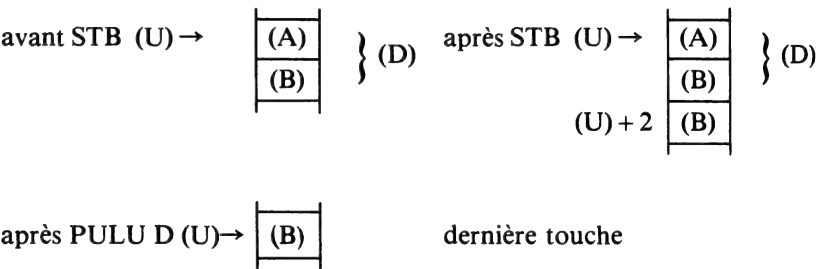
Nous préparons le retour en rangeant le dernier caractère, qui n'est pas un chiffre, à l'adresse (U) + 2 grâce à :

```
STB 2,U
```

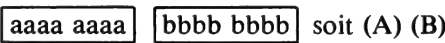
puisque cette instruction est suivie de :

```
PULU D
```

qui incrémente de 2 (U). Le schéma de la pile U est :



Le problème le plus délicat est la multiplication par 16 de (D), c'est-à-dire qu'il faut passer de :



à :



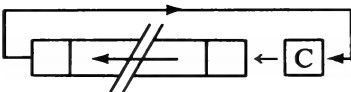
Il faut donc multiplier (B) par 2 — décalage à **gauche**, avec bit le poids fort dans C, et mise à 0 du bit de poids faible — et opérer un décalage à **gauche** dans (A) avec injection du bit de poids fort de (B) soit C. Ce qui est réalisé par :

```
ASLB Arithmetic Shift Left ou LSLB Logic SL  
ROLA Rotate Left
```

en effet le schéma de ASL est :



et celui de ROL :



Le programme de « construction » du nombre est écrit en sous-programme (il peut servir...), ce qui donne le listage suivant :

```

*  ENTREE DE NOMBRES AU CLAVIER
*  STOCKAGE EN MEMOIRE
*  LIMITES A 4 CHIFFRES

      E806   GETC   EQU   $E806
      E803   PUTC   EQU   $E803

6000 10CE 8000   DEBUT  LDS   $$8000
6004 CE    8100   LDU   $$8100
6007 8E    6C48   LDX   #TABLE
600A 108E 0000   LDY   #$0    COMPTEUR NBRE
600E 8D    6C1E   CHARGE JSR   CLAVIE  6 LETTRES !
6011 ED    81     STD   ,X++
6013 E6    C4     LDB   ,U
6015 31    21     LEAY  1,Y
6017 C1    D0     CMPB  #$0D    0D-30
6019 27    02     BEQ   SORTIE
601B 20    F1     BRA   CHARGE
601D 3F     SORTIE SWI

601E 4F     CLAVIE CLRA
601F 5F     CLRB
6020 36    06     ENTREE PSHU  D
6022 8D    E806   ENTRE  JSR   GETC
6025 5D     TSTB
6026 27    FA     BEQ   ENTRE
6028 8D    E803   JSR   PUTC
602B C0    30     SUBB  $$30
602D 2B    14     BMI   FIN
602F C1    09     CMPB  #$9
6031 22    10     BHI   FIN
6033 34    04     PSHS  B
6035 37    06     PULU  D      (D) = NOMBRE
6037 58     LSLB
6038 49     ROLA      (D) * 2
6039 58     LSLB
603A 49     ROLA      (D) * 4
603B 58     LSLB
603C 49     ROLA      (D) * 8
603D 58     LSLB
603E 49     ROLA      (D) * 16
603F EB    E0     ADDB  ,S+    REST. (S)
6041 20    D0     BRA   ENTREE
6043 E7    42     FIN    STB   2,U
6045 37    06     PULU  D
6047 39     RTS

6048 00     TABLE FCB  0

      6C00     END    DEBUT

```

16

SOUSTRACTION DECIMALE NOMBRES DE 2 CHIFFRES

BUT : *Réaliser une soustraction décimale.*

Problème : Comment réaliser une soustraction décimale sans utiliser des programmes de conversion (un pour passer du code DCB à l'hexadécimal avant d'effectuer l'opération — il sert 2 fois — et un pour traduire le résultat en DCB) étant donné que l'instruction DAS (équivalent de DAA pour la soustraction) n'existe pas.

Il faut comprendre comment s'opère une soustraction.

Pour soustraire un nombre d'une donnée, on... ajoute *son complément*. En binaire (base 2), il s'agit du complément à 2, en décimal il s'agit du complément à 10, 100...

Ainsi, en base 2, nous avons :

$$5 = 0101 \text{ et } 3 = 0011, \text{ ce qui donne } \bar{5}_1 = 1010, \bar{5}_2 = 1011, \bar{3}_1 = 1100 \text{ et } \bar{3}_2 = 1101$$

on obtient alors :

$$5 + \bar{5}_2 = 0000 \text{ et } 5 + \bar{3}_1 = 1111 \text{ de même pour 3 et ses compléments}$$

ce qui donne :

$$\begin{array}{r} 5 \quad 0101 \quad 3 \quad 0011 \\ -3 \quad + \quad 1101 \quad -5 \quad + \quad 1011 \\ \hline 2 \quad \boxed{1} \quad 0010 \quad -2 \quad \boxed{0} \quad 1110 \end{array}$$

En opérant « à la main », on constate que le résultat *positif* se manifeste par un dépassement de format égal à 1, et le résultat *négalif* par un dépassement égal à 0. Dans ce cas, le nombre est écrit sous la forme de son complément à 2 :

$$0010 + 1110 = 0000$$

En base 10, nous opérerons de même avec, si N est le nombre à soustraire :

$$\begin{aligned} \bar{N}_9 &= 9 - N \text{ pour un chiffre} \\ \bar{N}_{10} &= \bar{N}_9 + 1 \end{aligned}$$

ou pour un nombre de 2 chiffres :

$$\begin{aligned}\bar{N}_{99} &= 99 - N \\ \bar{N}_{100} &= \bar{N}_{99} + 1\end{aligned}$$

Par exemple :

$$\begin{aligned}70 - 36 &= 70 + ((99 - 36) + 1) \text{ soit} \\ &= 70 + 63 + 1 = \boxed{1} 34 \\ 36 - 70 &= 36 + ((99 - 70) + 1) \\ &= 36 + 29 + 1 = \boxed{0} 66\end{aligned}$$

Dans le premier cas, le résultat est *positif* — dépassement égal à 1 — dans le deuxième cas le résultat est *négalif*, et il vaut :

$$99 - 66 + 1 = 34 : \text{le résultat est } -34.$$

Nous écrivons $99 - N + 1$, et non $100 - N$, afin de préparer l'écriture du programme : 99 n'occupe que 8 bits. De plus, la soustraction de **n'importe quel** nombre de 99 donne un **résultat décimal**, sans retenue. Il suffira de penser à réaliser l'opération « + 1 » en décimal.

Dans le programme qui suit, le chargement des nombres est exécuté par programme, vous pouvez faire appel au programme 8. Nous mettons le résultat dans B avec le « signe » dans A : 0 = + et 1 = -.

```

#SOUSTRACTION DECIMALE 8 BITS
*   N1 - N2 = R
*R DANS B , SIGNE DANS A: 0=+

7000                                ORG    $7000

7000 86   71      DEBUT  LDA    #$71
7002 1F   88      TFR    A,DP      INIT. DP
7004 86   70      LDA    #$70      N1
7006 97   00      STA    $00
7008 86   36      LDA    #$36      N2
700A 97   01      STA    $01

      #DEBUT DU CALCUL

700C 0F   02      CLR    $02      SIGNE
700E 86   99      LDA    #$99
7010 90   01      SUBA   $01      99-N2
7012 88   01      ADDA   #1
7014 19      DAA          99-N2 +1 DEC.
7015 9B   00      ADDA   $00
7017 19      DAA          + N1 DEC.
7018 25   0B      BCS    FIN      C=1 > 0
701A 0C   02      INC    $02
701C 97   01      STA    $01
701E 86   99      LDA    #$99
7020 90   01      SUBA   $01
7022 88   01      ADDA   #1
7024 19      DAA
7025 1F   89      FIN    TFR    A,B
7027 96   02      LDA    $02
7029 3F      SWI

      7000      END    DEBUT

```

17

CONVERSION DCB-HEX POUR NOMBRES DE 16 BITS

BUT : *Conversion DCB-HEX pour nombres de 16 bits.*

Problème : Convertir un nombre décimal de 4 chiffres en hexadécimal.

Le programme que nous allons mettre au point peut être étendu à des nombres plus grands et servir de sous-programme.

Nous avons déjà vu que, pour convertir un nombre décimal (DCB) de 2 chiffres en hexadécimal, il fallait lui retrancher autant de fois 6 qu'il compte de dizaines. Pour traiter des nombres plus grands que 99, il suffit de remarquer que :

$$6 = 16 - 10$$

ainsi, nous devons retrancher :

$$256 - 100 = 156$$

autant de fois qu'il y a de centaines,

$$4096 - 1000 = 3096$$

autant de fois qu'il y a de milliers, et

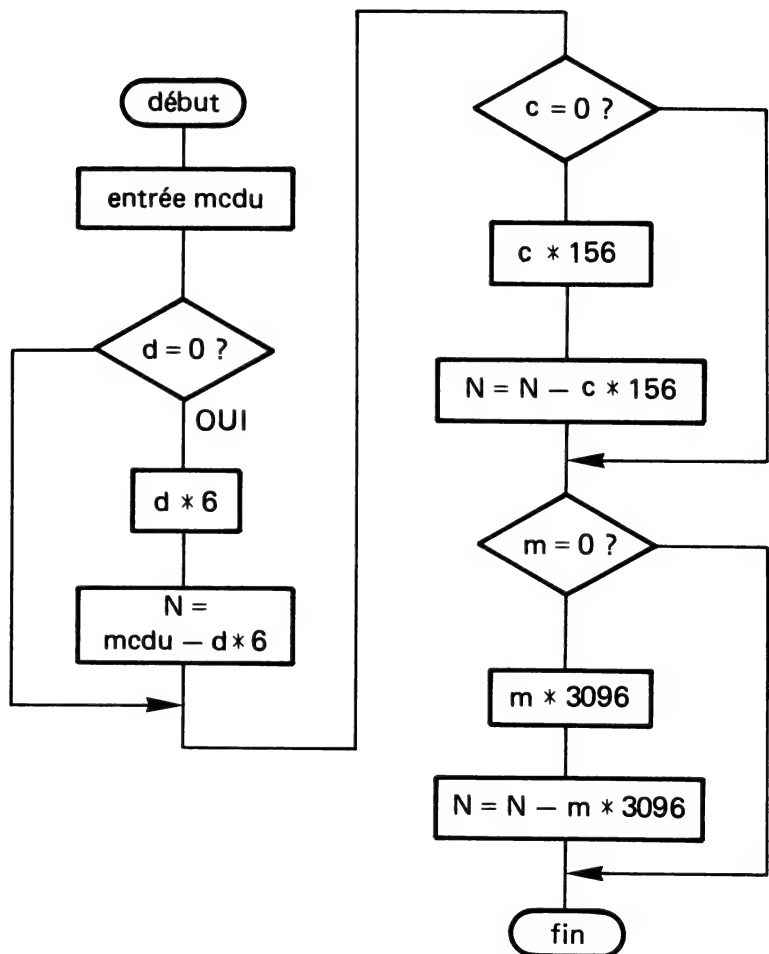
$$16 \text{ puissance } n - 10 \text{ puissance } n$$

pour chaque puissance n de 10.

Le premier problème est d'isoler dans un nombre « *mcd*u » (mille, cent, dix, unité) m , c , d et u . La solution est évidemment le *masquage* à l'aide de 0F ou F0 et de 4 rotations dans le cas de d et m .

Les opérations « $-6d$ » et « $-156c$ » peuvent être réalisées par des multiplications de 6 par d , et 156 par c — car 156 est plus petit que 255 — et une complémentation de (D) — voir programme 12. On opère la soustraction à l'aide de l'instruction :

LEAX D,X



Organigramme de la conversion DCB-Hexadécimal.

qui met dans X la somme : $(D) + (X)$.

Dans le cas de 3096 qui ne « tient » pas dans B, nous effectuons m fois l'opération commandée par :

LEAX - 3096,X

On obtient alors le listage suivant :

```

*CONVERSION DCB-HEX 16 BITS
*NOMBRE A CONVERTIR EN
*7100(POIDS FORT) ET 7101
*RESULTAT EN 7102 ET 7103 ET X

7000                                ORG      $7000

7000 86 71      DEBUT  LDA      #$71
7002 1F 88      TFR      A,DP
7004 86 99      LDA      #$99
7006 97 00      STA      $00
7008 86 99      LDA      #$99
700A 97 01      STA      $01      NBRE RANGE
700C 9E 00      LDX      $00

      *DEBUT DU TRAITEMENT
700E 96 01      LDA      $01      (A)=DIZ.UNITE
7010 84 F0      ANDA      #$0F0      DIZ.=0 ?
7012 27 0E      BEQ      SUITE0
7014 44          LSRA
7015 44          LSRA
7016 44          LSRA
7017 44          LSRA
7018 06 06      LDB      #$6
701A 3D          MUL
701B 43          COMA
701C 50          NEGB
701D 25 01      BCS      SUITE1
701F 4C          INCA
7020 30 8B      SUITE1 LEAX      D,X
7022 96 00      SUITE0 LDA      $00
7024 84 0F      ANDA      #$0F
7026 27 0A      BEQ      SUITE2
7028 06 9C      LDB      #156
702A 3D          MUL
702B 43          COMA
702C 50          NEGB
702D 25 01      BCS      SUITE3
702F 4C          INCA
7030 30 8B      SUITE3 LEAX      D,X
7032 96 00      SUITE2 LDA      $00
7034 84 F0      ANDA      #$0F0
7036 27 0B      BEQ      SUITE4
7038 44          LSRA
7039 44          LSRA
703A 44          LSRA
703B 44          LSRA
703C 30 89 F3E0 SUITE5 LEAX      -3096,X
7040 4A          DECA
7041 26 F9      BNE      SUITE5
7043 9F 02      SUITE4 STX      $02
7045 3F          SWI

      7000      END      DEBUT
```

Nous donnons une deuxième version qui comporte un sous-programme avec deux points d'entrée, ce qui permet de généraliser le procédé. On « entre » en SP0 pour les dizaines et les milliers, soit les quartets de poids fort, et en SP1 pour les quartets de poids faible (c...).

Pour arriver à ce résultat, on charge, si besoin, Y successivement à -6, -156 et -3096 ; Y servant de tampon car :

LEAX Y,X

n'existe pas et qu'il faut garder (A) qui vaut *d*, *c* ou *m*.

```

*CONVERSION DCB-HEX 16 BITS
*NOMBRE A CONVERTIR EN
*7100(POIDS FORT) ET 7101
*RESULTAT EN 7102 ET 7103 ET X
***AVEC SOUS-PROGRAMME

7000                                ORG    $7000

7000 10CE 8000    DEBUT  LDS    #$8000    INIT S
7004 86    71      LDA    #$71
7006 1F    88      TFR    A,D
7008 86    99      LDA    #$99
700A 97    00      STA    $00
700C 86    99      LDA    #$99
700E 97    01      STA    $01    NBRE RANGE
7010 9E    00      LDX    $00

      *DEBUT DU TRAITEMENT
7012 96    01      LDA    $01    (A)=DIZ.UNITE
7014 84    F0      ANDA    #$0F0    DIZ.=0 ?
7016 27    06      BEQ    SUITE0
7018 108E FFFA      LDY    #-6
701C 8D    18      BSR    SP0
701E 96    00      SUITE0 LDA    $00
7020 84    0F      ANDA    #$0F
7022 27    06      BEQ    SUITE1
7024 108E FF64      LDY    #-156
7028 8D    13      BSR    SP1
702A 96    00      SUITE1 LDA    $00
702C 84    F0      ANDA    #$0F0
702E 27    06      BEQ    SUITE2
7030 108E F3E8      LDY    #-3096
7034 8D    03      BSR    SP0
7036 9F    02      SUITE2 STX    $02
7038 3F                SWI

      *SOUS-PROGRAMME

7039 44                SP0    LSRA
703A 44                LSRA
703B 44                LSRA
703C 44                LSRA
703D 97    04      SP1    STA    $04    TAMPON
703F 1F    20      TFR    Y,D
7041 30    88      SP2    LEAX    D,X
7043 0A    04      DEC    $04
7045 26    FA      BNE    SP2
7047 39                RTS

      7000                                END    DEBUT

```

18

CONVERSION HEX-DCB POUR NOMBRES DE 16 BITS

BUT : *Convertir un nombre écrit en hexadécimal, comptant 4 quartets, en décimal.*

Problème : Convertir un nombre de 16 bits, écrit en hexadécimal, en décimal.

Le programme que nous allons mettre au point peut devenir un sous-programme servant à l'affichage de résultats.

Nous adopterons tout simplement le principe qui consiste à ajouter 16 au nombre des unités, préalablement converties, pour chaque « seizaine ».

La conversion des unités n'a lieu que si elles dépassent 9. En raison des valeurs des flags C et H après le test, nous utiliserons l'instruction :

DAA

Le problème est de mettre dans un registre le nombre des seizaines. Pour ce faire, il faut diviser le nombre par 16, afin de passer de :

mcd
à :
0mcd

Il faut donc :

diviser *du* par 16 pour arriver à *0d* à l'aide de :

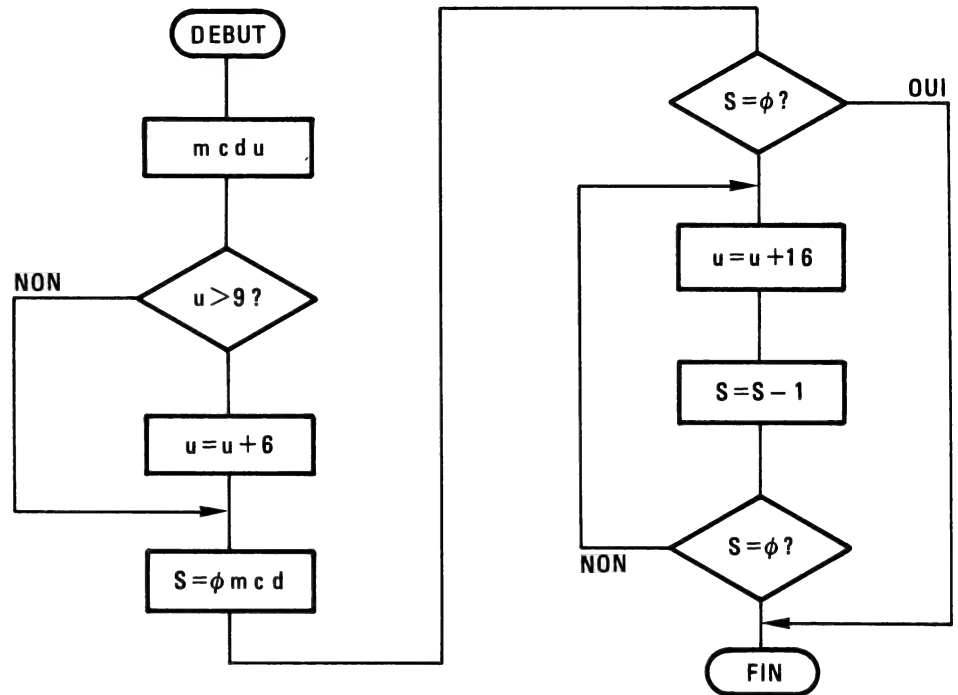
LSR *Logic Shift Right*
qui agit selon le schéma suivant :



- multiplier mc par 16 pour obtenir $c\theta$, afin de l'ajouter à θd , ce qui donne cd , à l'aide de :

ASL ou LSL

- diviser par 16 mc pour obtenir θm
- mettre θmcd dans un registre (16 bits) qui peut être décrémenté.



Organigramme de la conversion Hexadécimal-DCB

Nous ajouterons 16, en décimal, au nombre des « unités » en pensant à ajouter 1 pour chaque dépassement (voir programme 6) à chaque boucle ; le *décomptage* des boucles est obtenu par :

LEAX -1,X

qui affecte le flag Z.

Nous obtenons le listage suivant :

```

*CONVERSION HEX-DCB 16 BITS
*NOMBRE HEX. EN 7100,7101 (POIDS FORT)
*RESULTAT DEC EN 7102,7103
  
```

7000			ORG	\$7000	
7000	86	71	DEBUT	LDA	#\$71
7002	1F	8B		TFR	A,DP
7004	86	CD		LDA	#\$0CD OCTET "FAIBLE"
7006	97	00		STA	\$00
7008	86	AB		LDA	#\$0AB OCTET "FORT"
700A	97	01		STA	\$01 NBRE STOCKE

```

#DEBUT DU TRAVAIL...
7000 96 00          LDA      $00
700E 84 0F          ANDA     #$0F      TEST UNITES
7010 81 09          CMPA     #$09
7012 23 01          BLS      SUITE
7014 19            DAA
7015 97 02          STA      $02      SUITE
7017 0F 03          CLR      $03      MIL. CENT.=0
7019 96 00          LDA      $00
701B 44            LSRH
701C 44            LSRH
701D 44            LSRH
701E 44            LSRH      (A)=SEIZAINES
701F 97 05          STA      $05      TAMPON
7021 96 01          LDA      $01
7023 48            ASLA
7024 48            ASLA
7025 48            ASLA
7026 48            ASLA
7027 98 05          ADDA     $05
7029 97 05          STA      $05
702B 96 01          LDA      $01
702D 44            LSRH
702E 44            LSRH
702F 44            LSRH
7030 44            LSRH
7031 97 04          STA      $04      2EME TAMPON
7033 9E 04          LDX      $04      (X)="SEIZAINES"
7035 27 12          BEQ      FIN      (X)=0 ?
7037 96 02          LDA      $02      CONT
7039 88 16          ADDA     #$16
703B 19            DAA
703C 97 02          STA      $02
703E 96 03          LDA      $03
7040 89 00          ADCA     #0
7042 19            DAA
7043 97 03          STA      $03
7045 30 1F          LEAX     -1,X      (X)=(X)-1
7047 26 EE          BNE      CONT
7049 3F          FIN          SWI

7000          END          DEBUT

```

19

CONVERSION DCB-HEX POUR NOMBRES NON ENTIERS

BUT : *Convertir en hexadécimal la partie décimale d'un nombre.*

Problème : Traitement de la partie décimale — post virgule — d'un nombre.

Nous savons compter en base 2... les nombres « entiers ».

5483 contient 4096 1 fois plus
256 5 fois plus
16 6 fois plus
1 11 fois

soit $5483 = 156BH = 0001\ 0101\ 0110\ 1011B$

Mais comment traduire 0,5483 ?

Dans le cas des nombres entiers, nous utilisons les puissances positives de la base :

8 = 2 puissance 3 ($2 * 2 * 2$)
256 = 16 puissance 2 ($16 * 16$)
4096 = 16 puissance 3 ($16 * 16 * 16$)

Par contre, les nombres fractionnaires font appel aux puissances **négatives** comme 0,5 ou $\frac{1}{2}$ que l'on écrit :

2 puissance **moins** 1 ou 2^{-1}

Nous avons ainsi :

2 puissance moins 1 = 0,5
" 2 = 0,25
" 3 = 0,125
" 4 = 0,0625
" 5 = 0,03125
" 6 = 0,015625
" 7 = 0,0078125
" 8 = 0,00390625

Ainsi, 0,5483 est égal à :

$$\begin{aligned} &0,5 \quad \text{soit } 0.1 \\ &+ 0,03125 \quad \text{soit } 0.0000\ 1 \\ &+ 0,015625 \quad \text{soit } 0.0000\ 01 \\ &+ 0,001425 \quad \text{soit moins que } 0.0000\ 0001 \end{aligned}$$

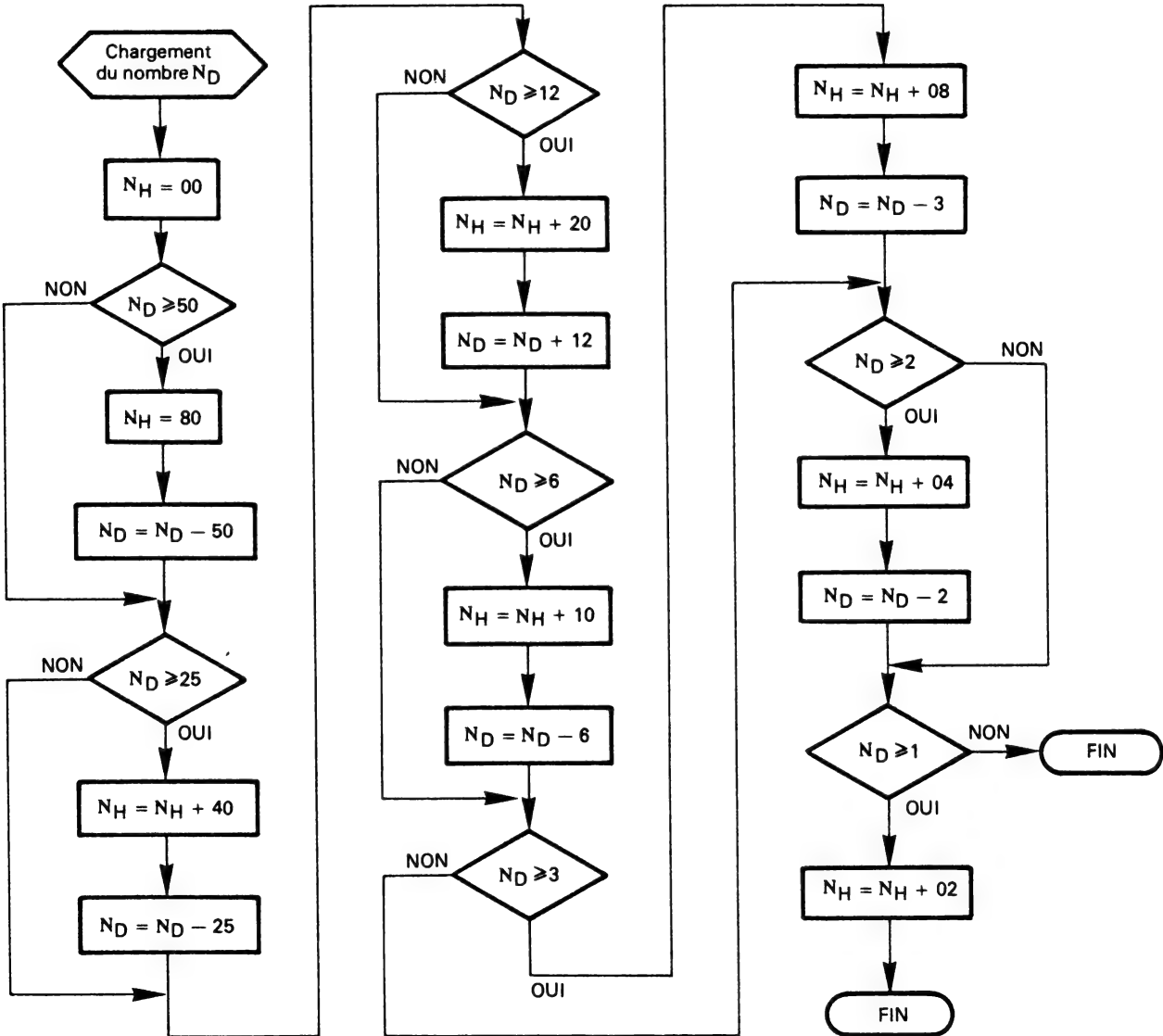
donc si nous nous limitons à 8 bits « décimaux », nous écrirons que :

$$0,5483 \text{ vaut } 0.1000\ 1100 \text{ soit } 0.8C$$

Le travail à exécuter consiste donc à voir si le nombre contient 0,5 ou 0,25 ou..., à chaque réponse affirmative on effectue la soustraction, on met le bit correspondant à 1 et on passe au test suivant.

Ce qui donne l’organigramme suivant :

Conversion DCB-hexadécimal



Organigramme de la conversion DCB-Hexadécimal pour nombres non entiers.

qui peut paraître compliqué, mais l'emploi d'une table simplifie le programme.

La table contient, arrondies à 2 chiffres, les 8 premières puissances négatives de 2. Le travail consiste à comparer le nombre à convertir à la première valeur de la table tout en préparant la comparaison suivante grâce à l'incréméntation du pointeur. Ce qui est obtenu par :

CMPA ,X+

si le nombre est plus petit (BLO) on teste si le nombre à convertir est nul ou si le nombre d'opérations à effectuer est nul.

Dans le cas où la soustraction est possible, on l'effectue en pensant que (X) est « en avance » d'une case, donc on écrit :

SUBA -1,X

Mais attention, la soustraction est *décimale* (voir le programme 16).
Le listage est le suivant :

```

*CONVERSION DCB-HEX POUR NOMBRES
*      NON ENTIERS
*RESULTAT DANS A

7000                                ORG      $7000

7000 10CE 8000      DEBUT  LDS      #$8000
7004 86   71        LDA      #$71
7006 1F   88        TFR      A,DP
7008 86   99        LDA      #$99      NBRE A CONV.
700A 97   00        STA      $00
700C 0F   01        CLR      $01      RESULTAT
700E 86   80        LDA      #$80      1000 0000B
7010 97   02        STH      $02
7012 8E   703B      LDX      #TABLE

                                #DEBUT DU TRAITEMENT
7015 96   00      OPER  LDA      $00
7017 A1   80        CMPA     ,X+
7019 25   08        BLO      SUITE
701B 8D   11        BSR      SOUS
701D 96   02        LDA      $02
701F 98   01        ADDA     $01
7021 97   01        STH      $01
7023 00   00      SUITE  TST      $0
7025 27   04        BEQ      FIN      NBRE NUL
7027 04   02        LSR      $02
7029 26   EA        BNE      OPER
702B 96   01      FIN   LDA      $01      POUR VOIR VITE
702D 3F                                SWI

```

```
702E 86 99      SOUS  LDA    ##99
7030 A0 1F      SUBA   -1,X
7032 8B 01      ADDA   ##1
7034 19        DAA
7035 9B 00      ADDA   $00
7037 19        DAA
7038 97 00      STA    $00
703A 39        RTS

703B 50 25 12 06 TABLE FCB    $50,$25,$12,$6,$3,$2,$1,$0
703F 03 02 01 00

          7000      END    DEBUT
```


20

CONVERSION HEX-DCB POUR NOMBRES NON ENTIERS

BUT : *Convertir la partie « décimale » d'un nombre hexadécimal.*

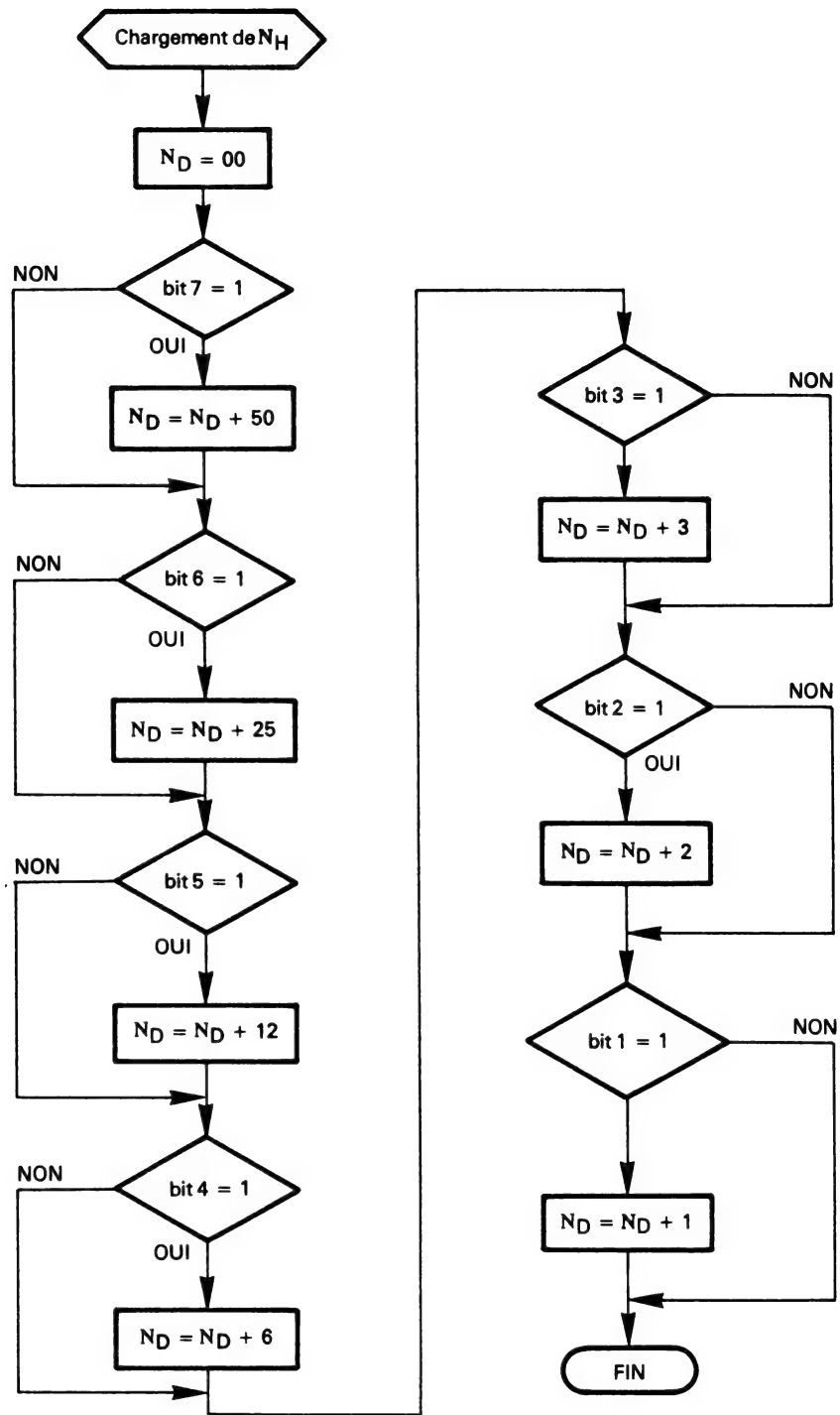
Problème : Traitement de la partie non entière d'un nombre hexadécimal.
Il s'agit du problème inverse du précédent, nous devons traduire, par exemple 0.8C en 0,55 (voir l'exemple du programme 19).

0.8C est en binaire 0.1000 1100, les différents bits égaux à 1 ont pour équivalent, respectivement :

0,5 0,03 et 0,02

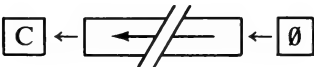
Le problème consiste donc à tester un à un les bits du nombre et sommer les valeurs décimales correspondantes en cas d'égalité à 1.

Ce qui donne l'organigramme suivant :



Organigramme de la conversion Hexadécimal-DCB pour nombres non-entiers.

Le test de la valeur des bits est assuré par un décalage à gauche avec copie du bit de poids fort dans C et mise à 0 du bit de poids faible, selon le schéma ci-dessous :



Cette opération est équivalente à une multiplication par 2 signée et est réalisée par l'instruction

ASL Arithmetic Shift Left

Les valeurs décimales des premières puissances négatives de 2 sont stockées en table (la même que pour le programme précédent).

Les additions successives n'ont lieu que si C est différent de zéro. Mais il faut avancer dans la table quand un bit est nul, c'est pourquoi nous n'utilisons pas l'adressage post-incrémenté.

La fin de la conversion est détectée par la valeur 0 pointée en table. On aurait pu tester l'arrivée à zéro du nombre à convertir après les multiplications par 2 successives.

Le listage obtenu est le suivant :

```

*CONVERSION HEX-DCB NOMBRE
*      "DECIMAL"

7000                                ORG      $7000

7000 86  71      DEBUT  LDA      #$71
7002 1F  88      TFR      A,DP
7004 86  7F      LDA      #$7F      NBRE A CONV.
7006 97  00      STA      $00
7008 8E  7021    LDX      #TABLE
700B 0F  01      CLR      $01      RESULTAT
700D 08  00      SUITE  ASL      $00      NBRE * 2
700F 24  07      BCC      ZERO      BIT A 0
7011 A6  84      LDA      ,X
7013 9B  01      ADDA     $01
7015 19          DAA
7016 97  01      STA      $01
7018 30  01      ZERO  LEAX     1,X      (X)=(X)+1
701A 6D  84      TST      ,X      ((X)) = 0 ?
701C 26  EF      BNE      SUITE
701E 96  01      LDA      $01
7020 3F          SWI

7021 50 25 12 06  TABLE  FCB      $50,$25,$12,6,3,2,1,0
7025 03 02 01 00

      7000      END      DEBUT
```

21

MULTIPLICATION HEXADÉCIMALE AVEC VIRGULE

BUT : Réaliser une multiplication hexadécimale non entière.

Problème : Lors de traitements mathématiques, les nombres rencontrés ne sont pas toujours entiers. Comment résoudre simplement le cas d'une multiplication ?

L'existence de l'instruction **MUL** simplifie le travail puisqu'elle retourne dans **D** le produit des contenus de **A** et **B**.

Pour obtenir un résultat facilement interprétable et aisément convertible en DCB, à l'aide des différents programmes que nous venons de voir, il suffit de définir la place des virgules pour que la partie non entière du produit soit toujours de 8 bits.

Ainsi, dans le cas de grandeurs physiques, il faudra choisir les unités. Par exemple, nous désirons multiplier l'intensité d'un courant par une tension pour obtenir une puissance :

$$\mathbf{P} = \mathbf{I} * \mathbf{U}$$

La puissance s'exprime en watt, compte tenu du nombre de bits de la partie non entière de P (8) le bit de poids faible représente :

2 puissance moins 8 watt

Il faut donc que le bit de poids faible de I représente :

2 puissance moins n ampère

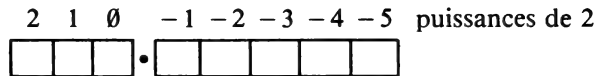
et celui de U :

2 puissance moins $(8 - n)$ volt.

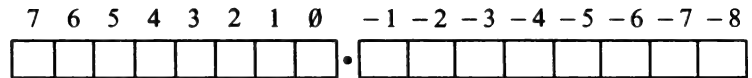
Nous pouvons prendre n égal à 3, ce qui donne une intensité au plus égale à 32 A car :

4	3	2	1	0	-1	-2	-3	Puissances de 2
<div style="border: 1px solid black; width: 30px; height: 30px;"></div>	<div style="border: 1px solid black; width: 30px; height: 30px;"></div>	<div style="border: 1px solid black; width: 30px; height: 30px;"></div>	<div style="border: 1px solid black; width: 30px; height: 30px;"></div>	<div style="border: 1px solid black; width: 30px; height: 30px;"></div>	.	<div style="border: 1px solid black; width: 30px; height: 30px;"></div>	<div style="border: 1px solid black; width: 30px; height: 30px;"></div>	<div style="border: 1px solid black; width: 30px; height: 30px;"></div>

Et la tension sera limitée à 8 volts :



La puissance est évidemment limitée à 256 watts



Nous aurons les différentes valeurs limites suivantes en fonction de la place de la virgule (*point*).

```

FF = 0.99 = 0.1111 1111
    1.99 = 1.111 1111
    3.98 = 11.11 1111
    7.96 = 111.1 1111
   15.93 = 1111. 1111
   31.87 = 1111 1.111
   63.75 = 1111 11.11
  127.50 = 1111 111.1
  255.00 = 1111 1111.
  
```

Si l'une des grandeurs doit être écrite avec 16 bits, c'est-à-dire si sa valeur maximale atteint :

```

FFFF = 65535.00 = 1111 1111 1111 1111.
32767.50 = 1111 1111 1111 111.1
16383.75 = 1111 1111 1111 11.11
8191.87 = 1111 1111 1111 1.111
4095.93 = 1111 1111 1111 .1111
2047.96 = 1111 1111 111.1 1111
1023.98 = 1111 1111 11.11 1111
511.99 = 1111 1111 1.111 1111
255.99 = 1111 1111 .1111 1111
  
```

nous devons écrire un programme qui multiplie 16 bits par 8 bits.

Pour cela, on sépare le mot de 16 bits en 2 de 8 bits *a* et *b*. Si le multiplicande est *c*, on calcule *a*c* d'une part et *b*c* d'autre part puis on ajoute *b*c* à *a*c* multiplié par 16 (un 0 à droite).

Ce qui donne le programme suivant :

```

* MULTIPLICATION HEXADÉCIMALE DE 16
*      BITS PAR 8 BITS
* LE MULTIPLICANDE EST EN 7100 (POIDS
* FORT) ET 7101 LE MULTIPLIATEUR EN
* 7102 ET LE RESULTAT EN 7103,7104,7105

7000                                ORG      $7000

7000 86    71      DEBUT  LDA      #$71
7002 1F    88              TFR      A,DP
7004 8E    74D7        LDX      #$74D7  POUR ESSAI(a b)
7007 9F    00          STX      $00
7009 86    C3          LDA      #$C3    c
700B 97    02          STA      $02
  
```

*debut du calcul

7000	96	00	LDA	\$00	(A) = a
700F	D6	02	LDB	\$02	(B) = c
7011	3D		MUL		
7012	00	03	STD	\$03	a*b
7014	96	01	LDA	\$01	(A) = b
7016	D6	02	LDB	\$02	(B) = c
7018	3D		MUL		
7019	D7	05	STB	\$05	
701B	9B	04	ADDA	\$04	
701D	97	04	STA	\$04	
701F	96	03	LDA	\$03	
7021	89	00	ADCA	#0	retenue eventuelle
7023	97	03	STA	\$03	
7025	3F		SWI		

* ici a= 74,b=D7,c=C3 soit ab =29911 et
* c = 195 le resultat est 58FFC5 soit
* 5832645

7000 END DEBUT

22

DIVISION HEXADÉCIMALE
16 BITS PAR 8 BITS
QUOTIENT NON ENTIER

BUT : *Réaliser une division.*

Problème : Comment faire une division en base 2, l'hexadécimal n'étant utilisé que pour le codage.

Une division en base 2 est une succession de tests et de soustractions comme en décimal, mais la création du quotient est bien plus simple puisque les chiffres utilisés sont 1 et 0.

Par exemple pour diviser 1011 1010 (186) par 0101 (5), on commence par écrire le complément à 2 du diviseur pour faciliter les opérations de soustraction. Soit, ici, 011 ou 1011 selon que l'on utilise 3 ou 4 chiffres (« tranche » de 3 ou 4 chiffres au dividende). On obtient donc :

3 chiffres

1011 1010

011

0001 10

0 11

0 0110

011

001000

4 chiffres

1011

00110

011

001000

0101

100101.00110011

la partie « décimale » du quotient est une suite infinie de 0011. Les retenues n'ont pas été figurées puisque lors d'une soustraction, on complémente le Carry.

On trouve que $186/5 = 37,19$ en utilisant la table du programme 20, au lieu de 37,2.

Dans le cas d'un microprocesseur, nous opérerons d'une façon plus simple en complémentant à 2 le diviseur étendu à 16 bits afin de le soustraire du dividende à chaque test positif et en incrémentant le quotient entier.

Ceci est possible grâce à l'instruction :

LEAX D,X

qui ajoute (D) à (X). Ici X contient le dividende et D le complément à 2 du diviseur.

Remarque. On pourrait penser qu'il suffit d'utiliser l'instruction :

LEAX A,X

puisqu'elle ajoute à (X) le contenu de A étendu, signé à 16 bits. Mais le complément à 2 du diviseur ne compte pas automatiquement un bit de poids fort égal à 1.

Par exemple, en se limitant à 4 et 8 bits :

le complément à 2 de 1010 est : 0110

ce qui donne, étendu signé à 8 bits : 0000 0110

alors qu'il nous faut le complément à 2 de 0000 1010 soit : 1111 0110 !

Le diviseur ne comptant que 8 bits et le dividende 16, le test s'effectue en 2 temps :

- l'octet de poids fort est-il différent de 0 ?
- si oui, on compare le dividende (ou le reste) au diviseur.

Dans le traitement de la partie non entière, nous nous limitons à 8 bits.

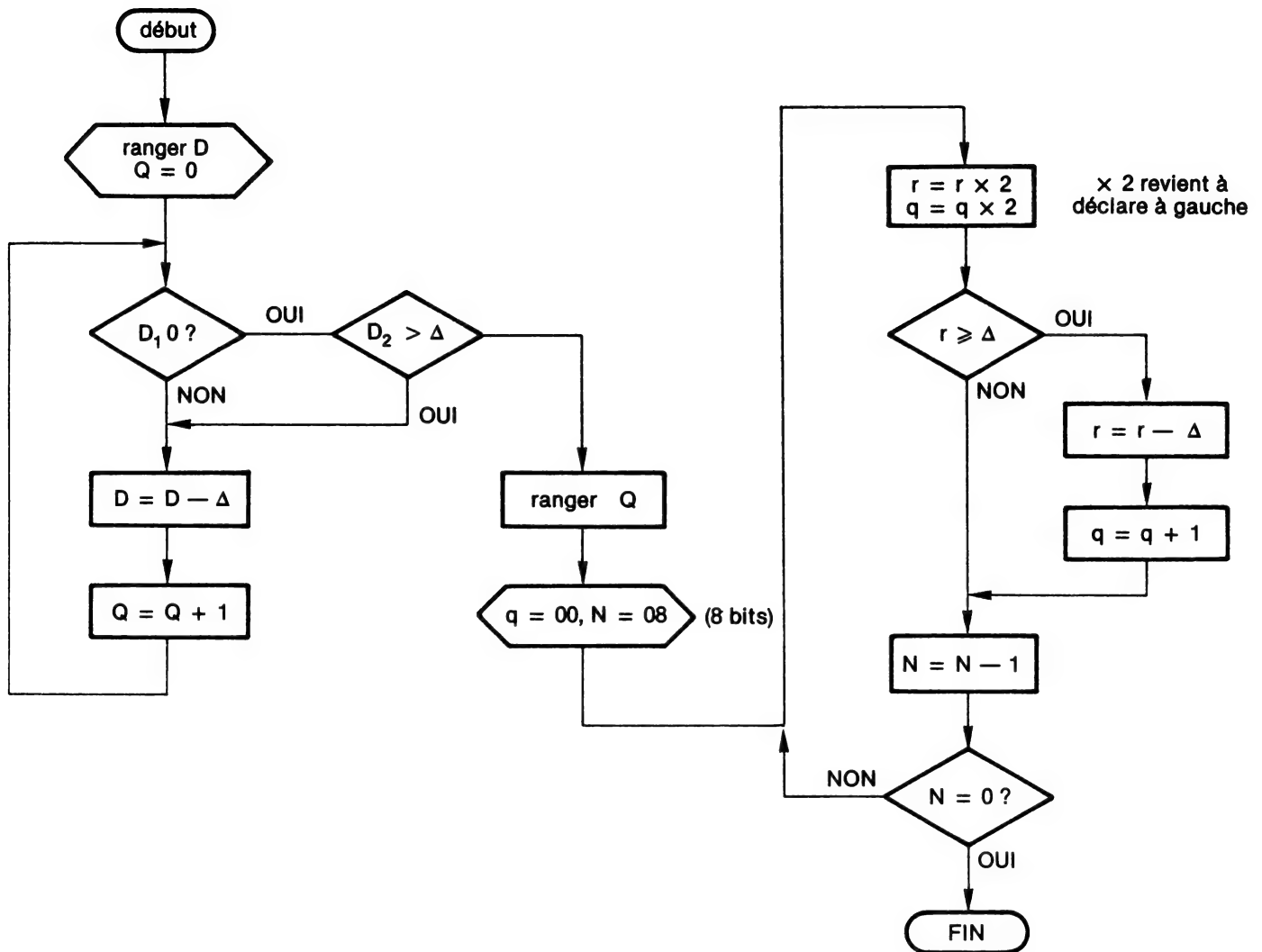
Pour opérer la division, il faut mettre un 0 à droite du reste, donc le multiplier par 2. S'il y a un Carry (*dépassement*), le nouveau nombre est évidemment plus grand que le diviseur, on ajoute 1 au quotient (q) qui a été préalablement multiplié par 2, puisque :

$$a/b = c \text{ et que } (2 * a)/b = 2 * c$$

Au bout de 8 multiplications, on a un 1 en bit de poids fort de q, si le premier test a été positif, ce qui est normal.

On obtient l'organigramme suivant, avec :

D = D₁ D₂ = dividende
 Δ = diviseur
 Q = quotient entier
 q = quotient non entier
 r = reste



Organigramme de la division.

Le listage obtenu est :

```

*DIVISION HEXADECIMALE 16 PAR 8 BITS
*   RESULTAT AVEC VIRGULE
*DIVIDENDE EN 7100 (POIDS FORT) ET
* 7101 OU X
*DIVISEUR EN 7102
*QUOTIENT EN 7103,7104 OU Y ET 7105
* OU A
*CASE 7106 CONTIENT LE COMPLEMENT A 2
* DU DIVISEUR

```

7000			ORG	\$7000	
7000	86	71	DEBUT	LDA	#\$71
7002	1F	88		TFR	A, DP
7004	86	0E		LDA	#\$0E
7006	97	00		STA	\$00
7008	86	86		LDA	#\$86
700A	97	01		STA	\$01
700C	9E	00		LDX	\$00
700E	86	A3		LDA	#\$0A3
7010	97	02		STA	\$02
7012	27	3E		BEQ	ERREUR
7014	40			NEGA	DIVISION PAR 0
7015	97	06		STA	\$06
7017	0F	03		CLR	\$03
7019	0F	04		CLR	\$04
701B	0F	05		CLR	\$05
701D	109E	03		LDY	\$03
7020	96	00	OPER	LDA	\$00
7022	26	08		BNE	SUITE0
7024	96	01		LDA	\$01
7026	27	27		BEQ	FIN
7028	91	02		CMPL	\$02
702A	25	11		BLO	VIRGUL
702C	31	21	SUITE0	LEAY	1, Y
702E	06	06		LDB	\$06
7030	86	FF		LDA	#\$0FF
7032	30	88		LEAX	D, X
7034	9F	00		STX	\$00
7036	20	E8		BRA	OPER
7038	109F	03		STY	\$03
703B	96	01		LDA	\$01
703D	06	08	VIRGUL	LDB	#\$08
703F	08	05	SUITE3	ASL	\$05
7041	48			ASLA	
7042	25	04		BCS	SUITE1
7044	91	02		CMPL	\$02
7046	25	04		BLO	SUITE2
7048	9B	06	SUITE1	ADDA	\$06
704A	0C	05		INC	\$05
704C	5A		SUITE2	DECB	
704D	26	F0		BNE	SUITE3
704F	96	05	FIN	LDA	\$05
7051	3F			SWI	
7052	3F		ERREUR	SWI	
7000			END	DEBUT	

23

PUISSANCE DÉCIMALE

BUT : Réaliser *a* puissance *b* en base 10.

Problème : Quel est l’algorithme de *a* puissance *b* ?

Dans le cas de la multiplication, nous avons :

$$P = a * b = \sum_1^b \underbrace{a = a + a + + a}_{b \text{ fois}}$$

Ici, il s’agit de :

$$E = a^b = a ** b = \prod_1^b \underbrace{a = a * a * ... * a}_{b \text{ fois}}$$

On remarque donc qu’il faut, si *b* est supérieur à 1, réaliser :

$$(((a * a) * a) * a...) * a$$

Nous nous limiterons à un résultat *E* de quatre chiffres, dans ce cas *a* et *b* sont plus petits que 10, puisque :

$$\begin{aligned} 2 ** 9 &= 512 \\ 3 ** 8 &= 6561 \\ 4 ** 6 &= 4096 \\ &\vdots \\ 9 ** 4 &= 6561 \text{ car } 9 = 3 ** 2 \text{ et} \\ (a ** b) ** c &= a ** (b * c) \end{aligned}$$

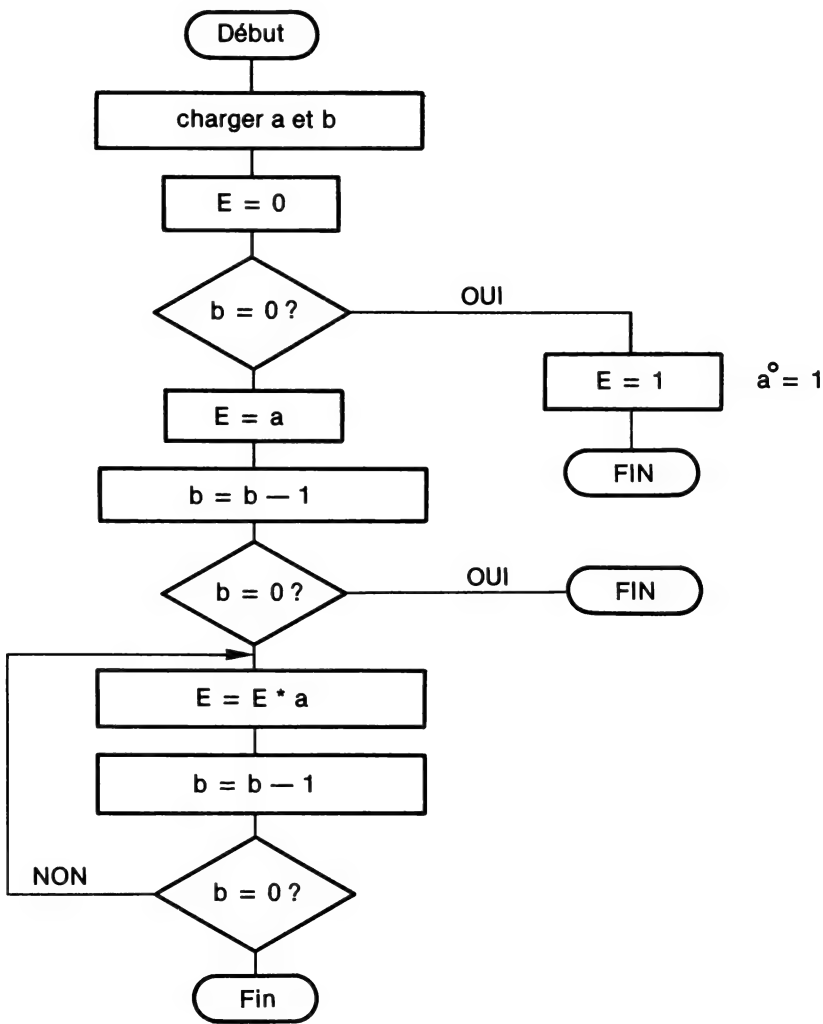
Nous pouvons utiliser le programme de la multiplication, mais à chaque parenthèse, le produit devient multiplicande :

$$E = E * a$$

Dans le cas de la multiplication, nous avons :

$$P = P + a$$

Le problème est que le multiplicateur doit « servir » plusieurs fois, il sera donc sau-
vegardé en pile — la sauvegarde d'un registre n'en détruit pas le contenu.
L'organigramme est :



Pour réaliser $E = E * a$, on stocke le résultat d'une multiplication (une opération entre parenthèses) en 7103, 7104 que l'on charge dans X afin de le transférer en 7100, 7101 cases contenant le multiplicande.

Le listage est alors le suivant :

```

*      P = a PUISSANCE b
*      DECIMALE a, b < 10
*      P < 9999 DANS X

7000                                ORG      $7000

7000 10CE 8000      DEBUT  LDS      #$8000
7004 86   71        LDA      #$71
7006 1F   88        TFR      A,DP
7008 0F   00        CLR      $0          POIDS FORT a
700A 86   04        LDA      #$04
700C 97   01        STA      $01
700E 9E   00        LDX      $00          (X) = 000A
7010 86   03        LDA      #$03          b
7012 97   02        STA      $02
7014 26   05        BNE      SUITE0      b # 0
7016 8E   0001      LDX      #$1          P = 1
7019 20   27        BRA      FIN
701B 0A   02        SUITE0 DEC      $02          b=1 -> P=a
701D 27   23        BEQ      FIN
701F D6   01        LDB      $01          (B) = b
7021 34   04        SUITE2 PSHS      B          SAUVE (B)
7023 0F   03        CLR      $03
7025 0F   04        CLR      $04
7027 96   04        SUITE1 LDA      $04
7029 9B   01        ADDA     $01
702B 19        DAA
702D 97   04        STA      $04
702F 96   03        LDA      $03          (03:04)=(00:01)*a
7031 99   00        ADCA     $00
7033 19        DAA
7035 97   03        STA      $03
7037 5A        DECB
7039 26   EF        BNE      SUITE1
703B 9E   03        LDX      $03          (X)=(03:04)
703D 9F   00        STX      $0          (00:01)=(03:04)
703F 35   04        PULS     B
7041 0A   02        DEC      $02          b = b - 1
7043 26   DF        BNE      SUITE2
7045 3F        FIN      SWI

7000                                END      DEBUT

```

24

PARITE

BUT : *Traitement de la parité des données.*

INSTRUCTIONS UTILISEES : *ROR, EOR.*

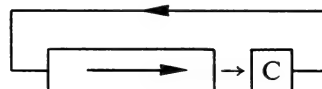
Problème : Tester la parité des données et « l'ajuster ».

La parité est une information essentiellement utilisée lors d'échange de données. Dans le protocole de communication, on **fixe** la parité des données échangées, si l'une d'elles n'a pas la *bonne* parité on émet un message d'erreur.
Qu'est-ce que la parité ? Il s'agit de la parité du **nombre de 1** d'un mot de 8 bits (en binaire 2 est impair et 3 est pair...).

Le code de transmission ASCII comporte 7 bits, on ajoute en bit de poids fort (bit 7) un 1 ou un 0 pour que la parité du mot soit **paire** (*even* — 4 lettres) ou **impair** (*odd* — 3 lettres : il s'agit de remarques mnémotechniques).

Il n'existe pas de test de parité chez 6809.

Nous devons donc compter les bits égaux à 1. Pour ce faire, nous effectuons des rotations de la donnée à tester selon le schéma suivant :



réalisées par l'instruction ROR (on peut préférer ROL) ; à chaque fois que le carry vaut 1, on incrémente un compteur de bits. Il faut effectuer 8 fois la séquence.

Le comptage des bits (1 + 1...) nous obligerait à effectuer un autre test... (N = 0, 2, 4, 6 ou 8 ?). Mais nous disposons du **ou-exclusif** (\oplus) qui donne :

$$\begin{array}{l} 0 \oplus 0 = 0 \\ 0 \oplus 1 = 1 \\ 1 \oplus 1 = 0 \end{array} \quad (\text{addition sans retenue})$$

donc si le nombre de bits 1 est pair le « compteur » de bits est nul, s'il est impair, le compteur de bits vaut 1.

L’instruction utilisée est :

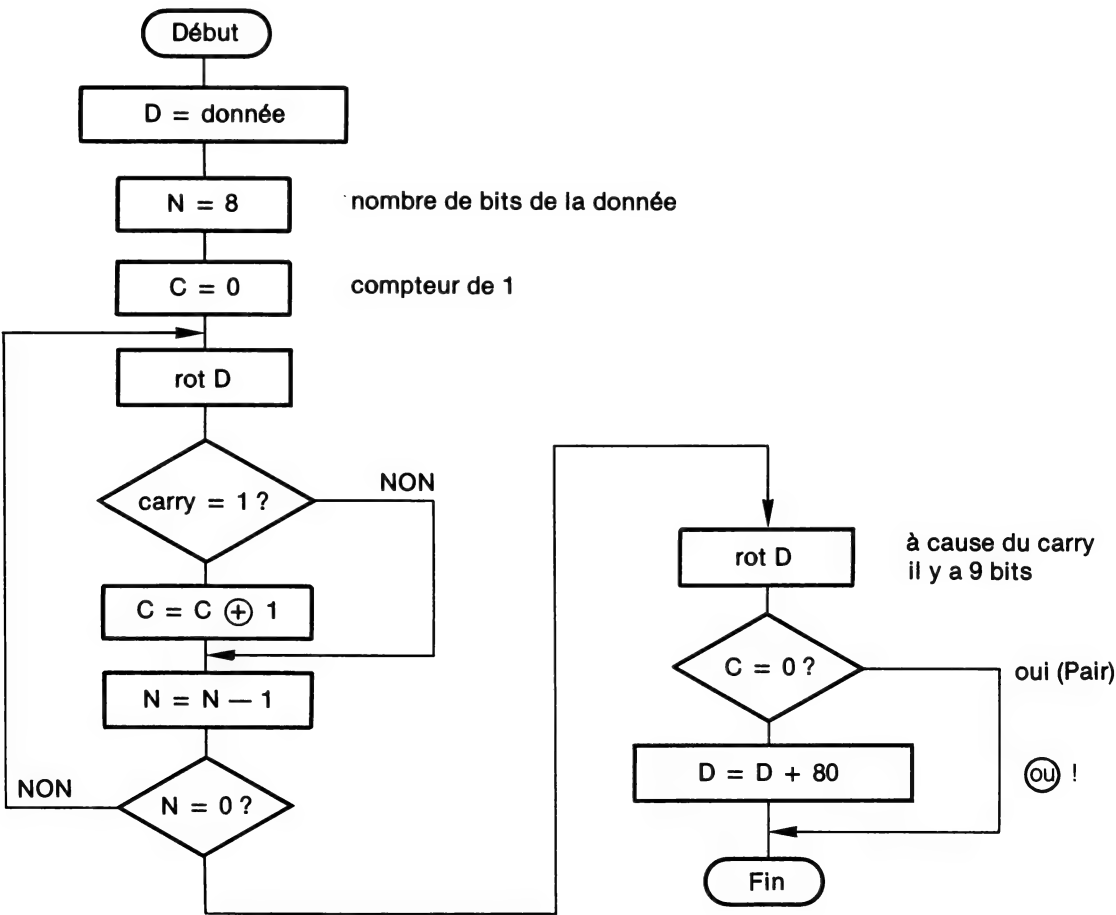
EOR

Si la donnée est paire, on ne fait rien, si elle est impaire, on ajoute un 1 en bit de poids fort à l’aide de l’instruction :

ORA #80 (80 = 1000 0000)

qui force le bit 7 à 1. (OU)

L’organigramme est le suivant :



Le programme est court, c'est pourquoi nous avons supprimé la directive ORG. Dans ce cas, le programme objet (en codes machine) commence au premier octet de la première page libre après le programme source (en mnémoniques) dont l'adresse est xy00.

* TEST DE PARITE					
6800	86	42	DEBUT	LDA	#'B ENTREE A TESTER
6802	87	7100		STA	\$7100
6805	06	08		LDB	##8 8 BITS
6807	4F			CLRA	(A) ET C = 0
6808	76	7100	SUITE	ROR	\$7100
680B	24	02		BCC	ZERO
680D	88	01		EORA	##1 (A) + 1
680F	5A		ZERO	DECB	
6810	26	F6		BNE	SUITE
6812	76	7100		ROR	\$7100 9 ROT. CAR CARRY
6815	4D			TSTA	
6816	27	08		BEQ	PAIR
6818	B6	7100		LDA	\$7100
6818	8A	80		ORA	##80 BIT PARITE
681D	B7	7100		STA	\$7100
6820	B6	7100	PAIR	LDA	\$7100 POUR TEST RAPIDE
6823	3F			SWI	
		6800	END	DEBUT	

25

AFFICHAGE AVEC TABLE DE CODES

BUT : *Adressage de données tabulées.*

INSTRUCTION UTILISEE : *ABX.*

Problème : Afficher des données quand les codes ASCII sont tabulés.

Le principe de la table est utile quand on travaille en hexadécimal car les chiffres décimaux sont codés de 30H à 39H, mais A est codé 41H.

On construit donc la table des codes :

'0'	30
.	.
'9'	39
'A'	41
.	.
'F'	46

Pour aller chercher le code du « digit » à afficher, il faut donc tout simplement charger un pointeur avec l'adresse du code de 0 par :

```
LDX #TABLE
```

qui met dans X l'adresse TABLE alors que :

```
LDX TABLE
```

met dans X le contenu des cases mémoires d'adresses TABLE et TABLE + 1.

Puis on ajoute au pointeur le digit à afficher de façon à pointer le code cherché ; si le digit est d, il faut réaliser (X) + d.

Ici, la table est limitée à 16 signes, si TABLE + 10H ne « saute » pas une page (256 octets) on utilise l'instruction :

```
ABX
```

qui ajoute à (X) le contenu de B non signé, sinon il faudra utiliser :

```
LEAX B,X
```

qui ajoute à (X) le contenu de B étendu, signé, à 16 bits. ((B) étant un digit est toujours positif).

On peut ainsi émettre un nombre très grand. Ici, on a fixé la fin du « message » par 00 et on a stocké le nombre à émettre dans la pile U (passage de paramètres), le programme commençant en AFF peut devenir un sous-programme.

Attention à l'ordre du stockage.

On utilise PULU pour vider la pile, mais on peut évidemment utiliser :

```
LDB ,U+ ou LDB , -U
```

*AFFICHAGE AVEC RECHERCHE EN TABLE
*LA TABLE CONTIENT LES CODES ASCII

```

      E803      PUTC, EQU      $E803
6B00 CE 7000      DEBUT      LDU      #$7000
6B03 10CE 8000      LDS      #$8000
6B07 6F C2      CLR      , -U
6B09 CC 0201      LDD      #$0201      POUR ESSAI
6B0C 36 06      PSHU      D
6B0E 8E 6B1F      AFF      LDX      #TABLE
6B11 37 04      PULU      B
6B13 5D      TSTB      POUR FLAGS !
6B14 27 08      BEQ      FIN
6B16 3A      ABX
6B17 E6 84      LDB      ,X
6B19 BD E803      JSR      PUTC
6B1C 20 F0      BRA      AFF
6B1E 3F      FIN      SWI

6B1F 30 31 32 33      TABLE      FCB      $30,$31,$32,$33,$34
6B23 34      FCB      $35,$36,$37,$38,$39
6B24 35 36 37 38
6B28 39
6B29 41 42 43 44      FCB      $41,$42,$43,$44,$45
6B2D 45
6B2E 46      FCB      $46

      6B00      END      DEBUT
```

26

CHOIX DE PROGRAMMES AU CLAVIER

BUT : *Appeler un programme particulier à l'aide d'une touche au clavier.*

INSTRUCTION UTILISEE : *JMP.*

ADRESSAGE : *Indexé Indirect.*

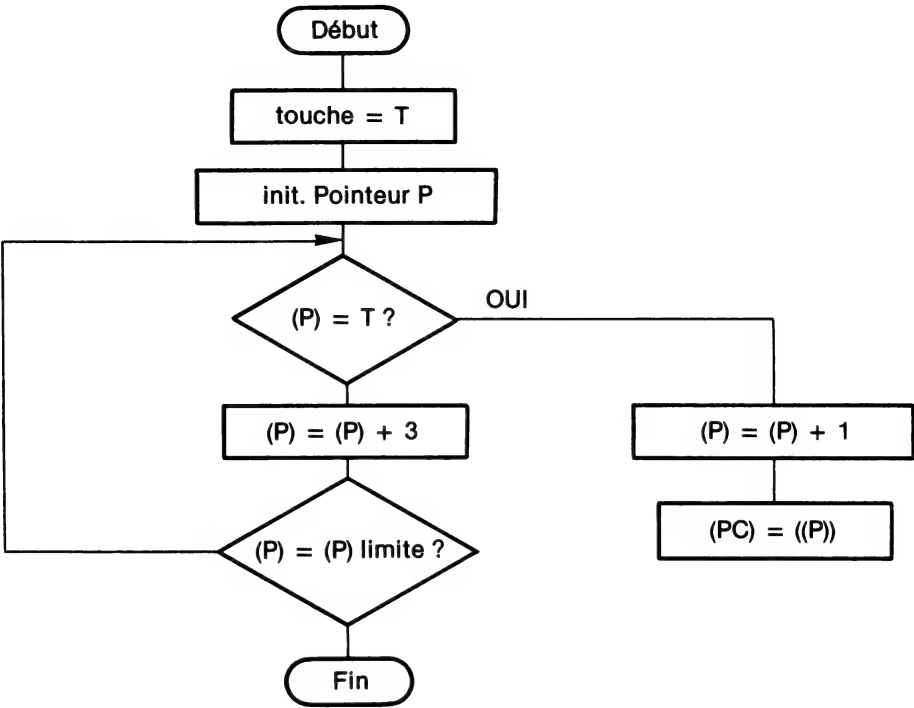
Problème : Différents programmes ont été stockés, indépendants les uns des autres, et on désire avoir accès à l'un d'entre-eux depuis une touche au clavier (voir votre Menu !).

Le principe est simple. On stocke en table les adresses de débuts de programmes. Si les codes d'appel ne diffèrent que d'une unité (0, 1, 2, 3... ou A, B, C, D..) — on peut éventuellement rebaptiser les touches — on double le code de la touche pressée, car une adresse occupe 2 octets, et on ajoute cette donnée à un pointeur de façon à pointer la case mémoire contenant l'adresse « haute » (octet de poids fort) du programme désiré. Il suffit alors de mettre cette adresse dans le compteur ordinal ou PC (*Program Counter*). Si les codes d'appel ne se suivent pas, il faut compléter la table précédente en faisant précéder chaque adresse du code de la touche d'appel. La suite est pratiquement identique au processus décrit plus haut.

Dans le premier cas, le pointage de l'adresse souhaitée s'effectue comme dans le programme précédent. Dans le deuxième cas, on effectue une série de comparaisons entre le code de la touche pressée et les codes stockés en mémoire, le pointeur est augmenté de 3 après chaque test négatif.

Il faut d'autre part prévoir un nombre limité de tests.

L'organigramme est donc :



La table est :

(P)	'A'	'A = 41H
(P) + 1		} adresse prog. A
(P) + 3	'B'	
(P) + 3 + 1		} adresse prog. B
(P) + 6		
(P) + 6 + 1		

Comment opérer (PC) = ((P)) ?

Cette question est commune aux deux processus décrits plus haut. On peut utiliser l'instruction TFR qui travaille avec PC, par exemple :

TFR U,PC

qui charge PC avec le contenu de U. On a fait précéder cette instruction de :

LDU ,X

si X est le pointeur, qui met dans U le contenu des cases mémoires pointées par X. Mais en utilisant l'adressage indexé **indirect** dont la syntaxe est, par exemple [Ø,X], on s'intéresse au contenu de la case mémoire dont l'adresse est le contenu de X.

Ainsi l'instruction :

JMP [Ø,X]

permet un saut (*Jump*) par chargement de PC avec le nombre pointé par le registre X.

Le listage est le suivant :

```

*RECHERCHE DE PROGRAMMES EN TABLE
*LE PROGRAMME EST APPELE GRACE A
*UNE LETTRE, ICI A,B ET C
*
* ADRESSAGE INDEXE-INDIRECT

7000                                ORG      $7000

7000 41                            TABLE  FCB      'A'
7001 7200                          FDB      PROGA
7003 42                            FCC      'B'
7004 7210                          FDB      PROGB
7006 43                            FCC      'C'
7007 7220                          FDB      PROGC

7009 00                            LIMITE  FCB      0          REPERAGE FIN

700A 8E      7000                  DEBUT   LDX      #TABLE
700D 06      02                    LDB      ##2
700F 86      41                    LDA      #'A'          POUR ESSAI
7011 A1      00                  SUITE    CMPA     ,X+
7013 27      07                    BEQ      TROUVE
7015 3A                                ABX                                (X) POINTE
                                                                LETTRE SUIVANTE
*
7016 8C      7009                  CMPX     #LIMITE
7019 26      F6                    BNE      SUITE
701B 3F                                SWI

701C 6E      98 00                  TROUVE  JMP      [0,X]

7200                                ORG      $7200

7200 3F                                PROGA   SWI

7210                                ORG      $7210

7210 3F                                PROGB   SWI

7220                                ORG      $7220

7220 3F                                PROGC   SWI

                                700A      END      DEBUT

```

On remarquera que l'on ajoute 3 en deux temps :

- par l'adressage indexé post-incrémenté de façon à pointer l'adresse en cas de test positif,
- en ajoutant 2 par ABX.

Les programmes n'ont pas été écrits, pour tester le bon fonctionnement on lit PC à chaque point d'arrêt, par exemple on aura :

pour le programme B : PC = 7210
si on tape D on aura PC = 701B.

27

TRI

BUT : *Ranger des nombres par valeur croissante ou décroissante. Manipuler l'adressage indexé.*

Problème : Une suite (*string*) de nombres est donnée dans le désordre. On désire les classer par ordre de valeur décroissante.

Le principe est simple : on effectue les permutations nécessaires et on incrémente un « *sémaphore* » — S — à chaque permutation réalisée. Il faut arriver à S égal \emptyset .

Par exemple, la suite initiale étant :

5 3 1 2 4

on compare 5 à 3, puis 3 à 1, puis 1 à 2 que l'on permute en incrémentant S initialement mis à \emptyset .

On obtient :

5 3 2 1 4

puis on compare 1 à 4, on permute et on incrémente S :

5 3 2 4 1 avec $S = 2$

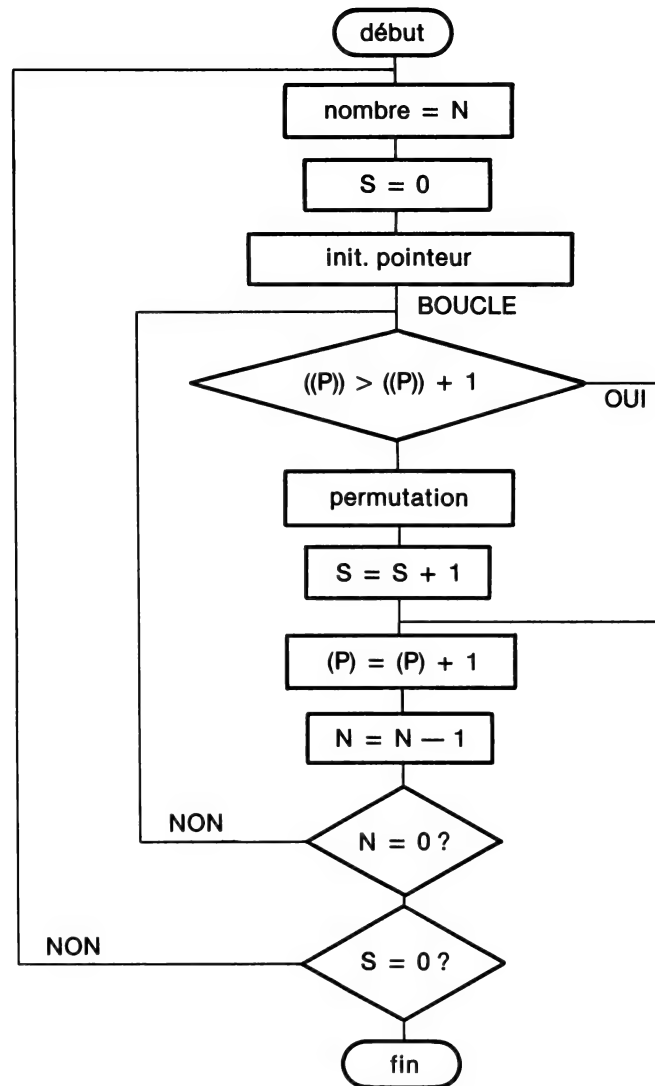
S étant différent de \emptyset , on aura successivement :

5 3 4 2 1 avec $S = 1$

5 4 3 2 1 avec $S = 1$

5 4 3 2 1 avec $S = \emptyset$

L'organigramme est :



Le problème réside dans le test :

$$((P)) > ((P) + 1)$$

Remarque : (P) est le contenu de P. ((P)) est le contenu de la case d'adresse (P). On charge A avec le contenu de la case pointée par X et on incrémente X pour pointer la case suivante :

LDA ,X+

il suffit alors d'écrire :

CMPA ,X

Pour permuter (échanger), on remarque que :

- A contient le plus petit nombre
- (X) pointe le plus grand

Il faut donc une case « tampon », on prend B que l'on charge avec le plus... grand :

LDB ,X

et on le range à la place du contenu de A soit :

STB -1,X

sans affecter X. Il reste à ranger (A) :

STA ,X

Le listage est donc :

					* TRI				
					*RANGEMENT DE NOMBRE PAR VALEUR				
					* DECROISSANTE				
7000				ORG	\$7000				
	0005	NBRE	EQU	5					
7000 86	71	DEBUT	LDA	#\$71					
7002 1F	88		TFR	A,DP					
7004 8E	7026	SUITE	LDX	#LISTE					
7007 86	05		LDA	#NBRE					
7009 97	00		STA	\$00					
700B 0F	01		CLR	\$01	INDIC. PERMUT.				
700D 0A	00	BOUCLE	DEC	\$00					
700F 27	10		BEQ	FINI					
7011 A6	80		LDA	,X+					
7013 A1	84		CMPS	,X	SUIVANT				
7015 24	F6		BHS	BOUCLE					
7017 E6	84		LDB	,X	ON PERMUTE				
7019 E7	1F		STB	-1,X					
701B A7	84		STA	,X					
701D 0C	01		INC	\$01					
701F 8D	EC		BSR	BOUCLE					
7021 0D	01	FINI	TST	\$01					
7023 26	DF		BNE	SUITE					
7025 3F			SWI						
7026 05	03 01 02	LISTE	FCB	5,3,1,2,4					
702A 04									
	7000	END	DEBUT						

28

DICTIONNAIRE

BUT : *Manipuler les instructions déjà vues. Manipuler l'adressage indexé.*

Problème : Chercher si un « *mot* » existe dans un « *dictionnaire* » sinon le ranger à la place qui lui convient.

Ce mot peut être l'*entrée* d'un « *fichier* ».

Pour une recherche efficace, il faut définir :

- la fin des mots,
- la fin de la suite de mots,
- la longueur maximale d'un mot (elle peut être celle du fichier appelé par le « mot »).

Nous avons choisi pour indiquer la fin d'un mot l'espace (20H en ASCII) et l'information NUL pour la fin du texte (00 en ASCII).

Il y a deux problèmes à résoudre. Le premier est la recherche alphabétique, le deuxième le rangement, souvent par insertion.

Recherche alphabétique

Il faut comparer les lettres une à une même si le processeur est de 16 bits.

Si la première lettre du mot à classer est identique ou pour la première fois « inférieure » (codage ASCII) à la première lettre d'un mot, on a *repéré* l'emplacement du mot. Il faut alors, s'il y a identité, comparer les lettres correspondantes qui suivent jusqu'à la *dernière* lettre du mot à classer, en se méfiant de mots stockés plus courts que le mot à ranger, puisque la fin indiquée par 20H est *inférieure* à toutes lettres (de 41H à 5AH pour les majuscules).

Par exemple avec BAR, BARBE et BAS, nous avons :

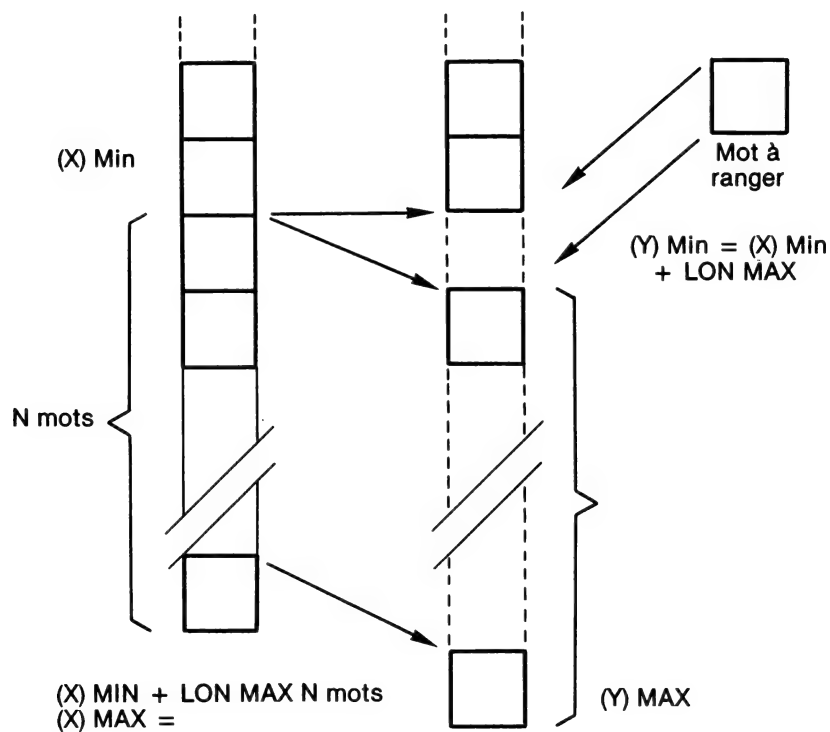
- BAR étant recherché et BARBE en table, le test arrêté à la fin du mot à ranger ferait croire qu'il existe déjà,
- BARBE étant recherché avec BAR et BAS en table, un test mal fait met BARBE **après** BAS.

Rangement

Pour ranger un mot dans une suite de mots, il faut :

- opérer un décalage libérant une zone mémoire intermédiaire,
- ranger le mot.

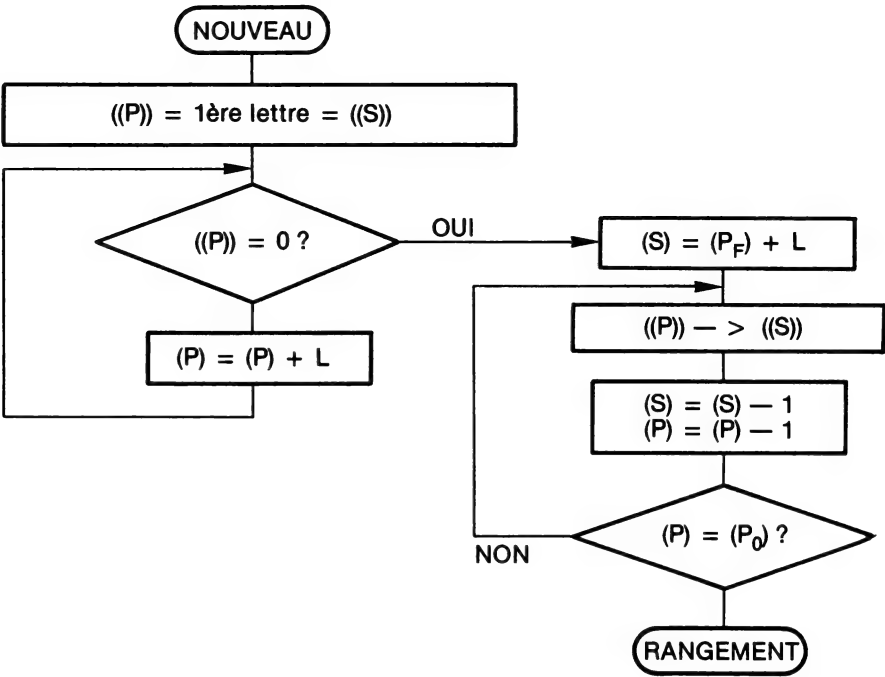
Le schéma des opérations est le suivant :



On remarque qu'aux sorties NOUVEAU, (X) pointe la première lettre du mot qui **doit** suivre dans la table finale le mot à stocker.

Il faut trouver le pointeur de la dernière lettre (00). Nous nous limitons à 65536 lettres au total (!). Il suffit d'incrémenter un pointeur tant que la case adressée ne contient pas 00.

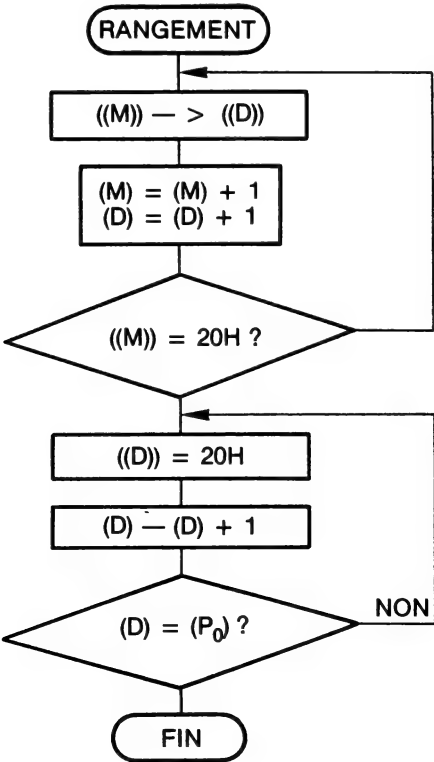
L’organigramme de cette opération est le suivant :



Le transfert est assuré par un adressage indexé prédécrémenté (les valeurs finales et initiales pour le rangement seront identiques : **prédécrémentées** et **postincrémentées**).

```
LDA    , - X
STA    , - Y
```

Puis on range le mot jusqu’à la donnée 20H et on complète la zone mémoire libérée par des espaces pour la « nettoyer ».



D'où le programme suivant :

```

*          DICTIONNAIRE
*LES MOTS STOCKES SONT POINTES PAR X
*LE MOT ENTRE PAR Y
*LA FIN D'UN MOT EST ' ' : 20ASCII
*LA PREMIERE CASE "VIDE" CONTIENT '00'
*LES MOTS ONT UNE LONGUEUR MAXIMALE :
* LONMAX.

```

7000		ORG	\$7000	
	0006	LONMAX EQU	06	
7000	0E	8000	DEBUT	LDU ##8000 FILE UTILISATEUR
7003	8E	7110		LDX #DICT
7006	108E	7102		LDY #MOT
700A	86	06		LDA #LONMAX
700C	5F			CLRB
700D	A6	A5	BOUCLE	LDA B,Y
700F	81	20		CMPA ##20
7011	26	06		BNE SUITE
7013	A1	85		CMPA B,X MOT PLUS LONG
7015	27	57		BEQ TROUVE
7017	20	20		BRA NEW
7019	A1	85	SUITE	CMPA B,X
701B	23	22		BLS REPERE (A) <= ((B)+(X))
701D	36	02		PSHU A
701F	A6	85		LDA B,X
7021	81	20		CMPA ##20 ((X)+(B)) = 20
7023	37	02		PULU A N'AFECTE PAS Z
7025	26	10		BNE LETSUI LEAX AFFECTE Z
7027	30	06		LEAX LONMAX,X
7029	36	02		PSHU A MOT SUIVANT PLUS
702B	A6	85		LDA B,X COURT ?
702D	81	20		CMPA ##20
702F	37	02		PULU A
7031	27	11		BEQ NEW
7033	A1	85		CMPA B,X MOT PLUS COURT
7035	23	08		BLS REPERE VOIR SUIVANT
7037	30	06	LETSUI	LEAX LONMAX,X
7039	6D	84		TST ,X
703B	26	DC		BNE SUITE
703D	20	05		BRA NEW A CAUSE FIN = 00
703F	26	03	REPERE	BNE NEW
7041	5C			INCB
7042	20	C9		BRA BOUCLE
7044	36	30	NEW	PSHU X,Y
7046	BF	7100		STX \$7100 CONSERVE (X) MIN
7049	6D	80	FIN	TST ,X+
704B	26	FC		BNE FIN RECH. FIN LISTE
704D	1F	12		TFR X,Y
704F	31	26		LEAY LONMAX,Y (Y) MAX.

```

7051 A6 82      DECAL LDA ,X
7053 A7 A2      STA ,Y
7055 BC 7100    CMPX $7100 (X) = (X) MIN ?
7058 26 F7      BNE DECAL
705A 10BF 7100  STY $7100 DERNIERE LETTRE
705E 37 30      PULU X,Y
7060 A6 A0      RANGT LDA ,Y+
7062 A7 00      ESP STA ,X+
7064 81 20      CMPA ##20
7066 26 F8      BNE RANGT
7068 BC 7100    CMPX $7100
706B 25 F5      BLO ESP

706D 3F         PLACE SWI          ICI MESSAGE
706E 3F         TROUVE SWI        ICI MESSAGE

7102          ORG $7102
7102 42 41 52 20 MOT FCC 'BAR '
7110          ORG $7110
7110 41 52 43 20 DICT FCC 'ARC BAC BAS BEAU '
7114 20 20 42 41
7118 43 20 20 20
711C 42 41 53 20
7120 20 20 42 45
7124 41 55 20 20
7128 42 45 40 40 FCC 'BELLE COUR '
712C 45 20 43 4F
7130 55 52 20 20
7134 0000      DER FDB $0
              7000      END DEBUT

```

A titre d'exemple, le dictionnaire contient ARC, BAC, BAS, BEAU, BELLE et COUR ; on a placé 2 points d'arrêt : un à NEW et un à RANGT.

- la recherche de BAR conduit à un arrêt en NEW, un en RANGT, où l'on voit deux fois BAS, et un en PLACE (fin),
- nous avons le même processus pour BA, BARBE et DAN, sans décalage,
- la recherche de BAR conduit à TROUVE.

A vous d'écrire les messages pour PLACE et TROUVE et le programme de chargement du mot à stocker. Vous vous inspirerez des programmes MESSAGE (9) et LISTE (12).

```

#D7102 7108
7102 42 41 52 20 FF 00 FF 00 BAR ....
#KNEW
#KRANGT
#D7110 7134
7110 41 52 43 20 20 20 42 41 ARC BA
7118 43 20 20 20 42 41 53 20 C BAS
7120 20 20 42 45 41 55 20 20 BEAU
7128 42 45 40 40 45 20 43 4F BELLE CO
7130 55 52 20 20 00 00 FF 00 UR ....
#GDEBUT
0 BRK @ NEW
#C
1 BRK @ RANGT
#D7110 7140
7110 41 52 43 20 20 20 42 41 ARC BA
7118 43 20 20 20 42 41 53 20 C BAS
7120 20 20 42 41 53 20 20 20 BAS
7128 42 45 41 55 20 20 42 45 BEAU BE
7130 40 40 45 20 43 4F 55 52 LLE COUR
7138 20 20 00 00 FF 00 FF 00 .....
7140 FF 00 FF 00 FF 00 FF 00 .....
#Y
#C
8 BRK @ PLACE
#D7110 7140
7110 41 52 43 20 20 20 42 41 ARC BA
7118 43 20 20 20 42 41 52 20 C BAR
7120 20 20 42 41 53 20 20 20 BAS
7128 42 45 41 55 20 20 42 45 BEAU BE
7130 40 40 45 20 43 4F 55 52 LLE COUR
7138 20 20 00 00 FF 00 FF 00 .....
7140 FF 00 FF 00 FF 00 FF 00 .....
#7102/ 42
#7103/ 41
#7104/ 52 20
#D7102 7108
7102 42 41 20 20 FF 00 FF 00 BA ....
#GDEBUT
8 BRK @ PLACE
#D7110 7140
7110 41 52 43 20 20 20 42 41 ARC BA
7118 20 20 20 20 42 41 43 20 BAC
7120 20 20 42 41 52 20 20 20 BAR
7128 42 41 53 20 20 20 42 45 BAS BE
7130 41 55 20 20 42 45 40 40 AU BELL
7138 45 20 43 4F 55 52 20 20 E COUR
7140 00 00 FF 00 FF 00 FF 00 .....
#7102/ 42
#7103/ 41
#7104/ 20 52
#7105/ 20 42
#7106/ 0FF 45
#7107/ 0 20
#D7102 7108
7102 42 41 52 42 45 20 FF 00 BARBE ..
#GDEBUT
8 BRK @ PLACE

```

```
#D7110 7140
7110 41 52 43 20 20 20 42 41 ARC BA
7118 20 20 20 20 42 41 43 20 BAC
7120 20 20 42 41 52 20 20 20 BAR
7128 42 41 52 42 45 20 42 41 BARBE BA
7130 53 20 20 20 42 45 41 55 S BEAU
7138 20 20 42 45 40 40 45 20 BELLE
7140 43 4F 55 52 20 20 00 00 COUR ..
```

```
#7102/ 42 44
```

```
#7103/ 41
```

```
#7104/ 52 4E
```

```
#7105/ 42 20
```

```
#D7102 7108
```

```
7102 44 41 4E 20 45 20 FF 00 DAN E ..
```

```
#GDEBUT
```

```
8 BRK @ PLACE
```

```
#D7110 7148
```

```
7110 41 52 43 20 20 20 42 41 ARC BA
7118 20 20 20 20 42 41 43 20 BAC
7120 20 20 42 41 52 20 20 20 BAR
7128 42 41 52 42 45 20 42 41 BARBE BA
7130 53 20 20 20 42 45 41 55 S BEAU
7138 20 20 42 45 40 40 45 20 BELLE
7140 43 4F 55 52 20 20 44 41 COUR DA
7148 4E 20 20 20 00 00 FF 00 N ....
```

```
#7102/ 44 42
```

```
#7103/ 41
```

```
#7104/ 4E 52
```

```
#7105/ 20
```

```
#D7102 7108
```

```
7102 42 41 52 20 45 20 FF 00 BAR E ..
```

```
#GDEBUT
```

```
8 BRK @ TROUVE
```


29

AFFICHAGE SEMI-GRAPHIQUE

BUT : Apprendre à gérer l'écran en mode semi-graphique.

Avant de traiter le problème de l'affichage semi-graphique, nous pouvons écrire un court programme qui nous donne tous les caractères codés en ASCII :

```
PUTC EQU $E803
DEBUT LDB #$20 code 'espace'
AFF JSR PUTC
      INCB
      CMPB #$80 le dernier ASCII + 1
      BNE AFF
      SWI
```

nous aurons à l'écran tous les caractères codés de 20H à 7FH, ce dernier (DEL) étant un pavé.

La technique de création d'un caractère consiste en l'allumage de points d'une matrice qui en compte 64 (8*8), par exemple pour A (1 = allumé).

```
0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0
0 0 1 0 0 1 0 0
0 1 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0
```

ce qui donne 8 octets par caractère, ici :

00, 42H, 42H, 7EH, 42H, 24H, 18H et 00

Puisque le **premier octet** utilisé est celui du **bas**,

Nous pouvons, selon le même principe, créer nos propres « caractères » qui seront « codés ASCII » de 80H à FFH, à condition de stocker dans un « registre » de l'ordinateur, baptisé USERAF (USER AFFichage) l'adresse du premier octet du caractère codé 80H.

Ainsi la table qui suit le programme permet d’afficher à la suite des caractères classiques :



A vous de générer vos propres caractères.

Pour placer ces diverses figures en des points bien précis de l’écran, il suffit de positionner le curseur à l’endroit désiré en émettant, grâce à PUTC : 1F suivi de la position Y, de 40H à 58H (0 à 24) et de la position X, de 41H à 68H (1 à 40). Par exemple pour mettre le curseur en X = 16 et Y = 8, on écrit :

```
LDB #$1F
JSR PUTC
LDB #$48      Y
JSR PUTC
LDB #$50      X
JSR PUTC
```

* TEST DES CODES CARACTERES									
* ETENDU									
		E803	PUTC	EQU	\$E803				
		602D	USERAF	EQU	\$602D				
6B00	10CE	8000	DEBUT	LDS	#\$8000				
6B04	8E	6B15		LDX	#MESCAR				
6B07	BF	602D		STX	USERAF				
6B0A	C6	20		LDB	#\$20				
6B0C	BD	E803	AFF	JSR	PUTC				
6B0F	5C			INCB					
6B10	C1	85		CMPB	#\$85	LIMITE			
6B12	26	F8		BNE	AFF				
6B14	3F			SWI					
6B15	18	18	18	FF	MESCAR	FCB	\$18,\$18,\$18,\$FF	#	
6B19	FF	18	18	18		FCB	\$FF,\$18,\$18,\$18		
6B1D	FF	80	80	80		FCB	\$FF,\$80,\$80,\$80	└	
6B21	80	80	80	80		FCB	\$80,\$80,\$80,\$80		
6B25	80	80	80	80		FCB	\$80,\$80,\$80,\$80	┐	
6B29	80	80	80	FF		FCB	\$80,\$80,\$80,\$FF		
6B2D	01	01	01	01		FCB	1,1,1,1,1,1,1,\$FF	┘	
6B31	01	01	01	FF					
6B35	FF	01	01	01		FCB	\$FF,1,1,1,1,1,1,1	┘	
6B39	01	01	01	01					
		6B00		END	DEBUT				

30

GRAPHISME

BUT : *Apprendre à utiliser le photostyle et gérer les couleurs.*

Problème : Le graphisme.

Un dessin sur écran est constitué d'une succession de points de couleur. Il faut donc pouvoir « allumer » un point en lui donnant une certaine couleur. Un point « allumé » correspond à un bit à 1 (voir programme 29) ; un point « éteint » peut être de couleur, le bit correspondant est à 0, et constitue le FOND. Si le FOND est noir, il faudra pour avoir des points de couleur :

- mettre un code POSITIF de 0 à 15 pour TO7-70 et de 0 à 7 pour TO7, dans le « registre » FORME (un code négatif donne un point « éteint » au sein d'un segment « allumé » de 8 points),
- mettre à zéro le contenu du « registre » CHDRAW (graphisme),
- appeler le programme PLOT après avoir chargé X et Y aux coordonnées du point (0 à 319 et 0 à 199).

Ainsi pour allumer en magenta un point en 160/100 (milieu de l'écran), on écrit :

PLOT	EQU	\$E80F	
FORME	EQU	\$6038	
CHDRAW	EQU	\$6041	
DEBUT	CLR	CHDRAW	
	LDS	# \$8000	
	LDX	# 160	
	LDY	# 100	
	LDA	# 5	MAGENTA
	STA	FORME	
	JSR	PLOT	
	SWI		
	END	DEBUT	

Pour tracer un segment de droite, on appelle le programme DRAW qui trace, entre le dernier point allumé par PLOT (ou DRAW) et le point dont les coordonnées sont « passées » par X et Y.

Si l'on désire utiliser un photostyle pour dessiner sur l'écran, il faut :

- tester la pression du crayon sur l'écran en appelant le programme LPIN, qui met le carry à 1 si le crayon est appuyé,
- appeler le programme GETL qui retourne les coordonnées du point visé dans X et Y, avec C à 1 si le point est hors page (0,319 - 0,199) ou peu lumineux.

En raison du dernier point, il faut viser une zone claire et délimiter la page.

Pour ce faire, on donne une couleur au tour différente de celle du fond, en utilisant une séquence dite « d'échappement », pour modifier **tout** l'écran par rapport à la version moniteur (lettres claires sur fond noir), et une modification du type « courant » pour le tour.

Ce qui donne la série de codes à envoyer suivante :

- pour le fond : 1B (ESCape), 23 (#), 20 (), 56 (fond cyan),
- pour le tour : 1B, 60 (tour noir)
complétée, pour « nettoyer » l'écran, par le code 0C. (les codes sont évidemment donnés en hexadécimal).

Pour éviter des égarements au 6809, on teste l'état du carry après l'appel à LPIN, et les contenus de X et Y ne sont conservés après l'appel à GETL que si le carry est nul.

En raison de la rapidité du système, on a mis une TEMPORISATION obtenue par une boucle qui décompte jusqu'à zéro le contenu de Y.

On démarre avec la couleur magenta sur fond cyan et on n'utilise que 8 couleurs. Tant que l'on n'appuie pas sur une touche chiffrée de 0 à 7, on garde la couleur précédente ; une touche chiffrée appuyée change la couleur du trait.

Les touches 6 et 7 ne donnent pas de trait (couleur du fond ou blanc).
La touche 8 efface le dessin.

Comme le « fond » est constitué de points « éteints » (bit à 0), nous devons « allumer » (bit à 1) les points du dessin. Pour cela, le code des couleurs est POSITIF (de 0 à 7 ou 15).

Soyez créatifs !...

```

                                *ESSAI GRAPHISME AU PHOTOSTYLE

                                E01B    LPIN    EQU    $E01B
                                E018    GETL    EQU    $E018
                                E006    GETC    EQU    $E006
                                E00F    PLOT    EQU    $E00F
                                6038    FORME    EQU    $6038
                                6041    CHDRAW EQU    $6041
                                E00C    DRAW    EQU    $E00C
                                E003    PUTC    EQU    $E003

0C00 10CE 0000    DESSIN LDS    $$0000
0C04 8E    6C4D          LDX    #MESSAGE
0C07 8D    6C43          JSR    INIT
0C0A 7F    6041          CLR    CHDRAW GRAPHISME
0C0D 86    05           LDA    #05     FORME
0C0F 87    6038          STA    FORME
0C12 8D    E81B    STYLO JSR    LPIN
0C15 24    FB           BCC    STYLO
0C17 8D    E818          JSR    GETL
0C1A 8D    E80F          JSR    PLOT
0C1D 10BE 0000    TEMPS LDY    $$0000
0C21 31    3F          TEMPO LEAY -1,Y
0C23 26    FC           BNE    TEMPO
```

```

6025 B0 E81B GRAPH JSR LPIN
6028 24 FB BCC GRAPH
602A B0 E81B JSR GETL
602D 25 EE BCS TEMPS
602F B0 E806 JSR GETC
6032 5D TSTB
6033 27 09 BEQ SUITE
6035 C0 30 SUBB #$30
6037 C1 08 CMPB #$8
6039 24 C5 BHS DESSIN
603B F7 6038 STB FORME
603E B0 E80C SUITE JSR DRAW
6041 20 DA BRA TEMPS

6043 E6 80 INIT LDB ,X+
6045 27 05 BEQ FIN
6047 B0 E803 JSR PUTC
604A 20 F7 BRA INIT
604C 39 FIN RTS

604D 1B 23 20 56 MESSAGE FCB $1B,$23,$20,$56,$0C
6051 0C
6052 1B 60 00 FCB $1B,$60,0

        6000 END DESSIN

```

31

GRAPHISME MATHÉMATIQUE SINUSOÏDE

BUT : *Tracer des courbes, gérer une table.*

Problème : Tracer une période de sinusoïde.

Nous avons vu qu'il est très simple de tracer des segments de droites à l'aide de DRAW, mais comment tracer des courbes ?

A titre d'exemple, nous avons choisi de tracer une sinusoïde. Le premier problème à résoudre est donc le calcul du sinus d'un angle. Les mathématiciens vous diront que l'on peut calculer le sinus à l'aide de la formule :

$$\sin \theta = \theta - \theta^3/6 + \theta^5/120 \dots \text{ où } \theta \text{ est en radians.}$$

Il est hors de notre propos de mettre au point un programme donnant le sinus de n'importe quel angle avec 9 chiffres après la virgule, puisque le sinus est compris entre 1 et -1.

Dans notre cas, le problème est de tracer une sinusoïde à l'écran. Or, les coordonnées des points sont **entières**. Nous multiplierons donc le sinus par une constante inférieure à 100, puisque l'écran admet 200 points en verticale.

Nous désirons tracer *une période* de sinusoïde, c'est-à-dire porter les valeurs que prend le sinus d'un angle compris entre 0 et 360 degrés.
Pour **éviter le calcul** du sinus, nous... écrirons une **table**.

Afin de limiter la longueur de la table, on remarque que les valeurs du sinus sont identiques, au signe près, pour les angles de 0 à 180 degrés et les angles de 180 à 360 degrés. La table ne concernera que les angles de 0 à 180 et le programme sera composé de deux parties suivant le signe du sinus.

Comme le sinus est une fonction lentement variable, nous prendrons dans un livre les valeurs pour les angles 0, 5, 10... et nous traduirons les nombres décimaux en hexadécimal (programme 19), la virgule étant placée à gauche du bit de poids fort.

Ainsi, le résultat du produit du sinus par la constante est composé de 2 octets... séparés par une virgule (programme 21), dont nous ne garderons que l'octet de poids fort.

La table des valeurs décimales et hexadécimales est la suivante :

angle	sinus décimal	sinus hexadécimal
0	0	00
5	0,09	.18
10	0,17	.2A
15	0,26	.42
20	0,34	.56
25	0,42	.6C
30	0,50	.80
35	0,57	.92
40	0,64	.A4
45	0,70	.B4
50	0,77	.C4
55	0,82	.D2
60	0,87	.DE
65	0,91	.EA
70	0,94	.F2
75	0,96	.F6
80	0,98	.FC
85	0,99	.FF
90	1	.FF
95	0,99	.FF
.	.	.
.	.	.
180	0	00

Les valeurs sont identiques entre 90 et 180 degrés à celles entre 90 et 0 degrés. Nous pourrions limiter la table mais cela compliquerait inutilement le programme.

Pour tracer la courbe, il faut adopter une échelle sur l'axe X. Nous disposons de 320 points pour 360 degrés ! En partant de X=0, si nous augmentons X de 4 pour un saut de 5 degrés, nous aurons une période de l'abscisse 0 à l'abscisse 292 ((360/5)*4 + 4).

En choisissant pour origine Y=100 (64H) et X=0, et pour constante 50 (32H), nous devons calculer pour les angles de 0 à 180 degrés :

$$Y = 100 - 50 * \sin \theta$$

et pour les angles de 180 à 360 degrés :

$$Y = 100 + 50 * \sin \theta$$

Pour ce faire, on utilise U comme pointeur de la table des sinus avec l'adressage indexé post-incrémenté jusqu'à la valeur 0 du sinus, puis avec l'adressage indexé prédécémenté jusqu'à la valeur 0 du sinus.

Il faut éviter, en raison des tests d'égalité à 0 :

- l'interruption prématurée du programme : on porte le point 0,100 en premier et on pointe l'angle 5 degrés,
- la répétition du point 180 degrés : on décrémente (U) et on ajoute 4 à (X) avant de commencer le calcul pour la partie « négative » de la courbe.

Le listage est le suivant :

```

                                *ESSAI GRAPHISME MATHEMATIQUE
                                *      SINUSOÏDE

                                6038  FORME EQU  $6038
                                6041  CHDRAW EQU  $6041
                                E80C  DRAW EQU  $E80C
                                E803  PUTC EQU  $E803
                                E80F  PLOT EQU  $E80F

0000 10CE 8000  DESSIN LDS  #$8000
0004 8E 606A    LDX  #MESSAGE
0007 8D 6060    JSR  INIT
000A 7F 6041    CLR  CHDRAW GRAPHISME
000D 8E 0000  HOR  LDX  #0      HORIZONTALE
0010 108E 0064    LDY  #$64
0014 7F 6038    CLR  FORME NOIRE
0017 8D E80F    JSR  PLOT
001A 8E 013F    LDX  #319
001D 8D E80C    JSR  DRAW
0020 86 05     LDA  #5      FORME
0022 B7 6038    STA  FORME MAGENTA
0025 8E 0000    LDX  #0
0028 108E 0064    LDY  #$64
002C 8D E80F    JSR  PLOT
002F CE 6D73    LDU  #TABLE+1
0032 E6 C0     POS  LDB  ,U+
0034 86 32     LDA  #$32
0036 30 04     LEAX 4,X
0038 3D        MUL
0039 40        NEGA
003A 108E 0064    LDY  #$64
003E 31 A6     LEAY  A,Y
0040 8D E80F    JSR  PLOT
0043 6D 5F     TST  -1,U
0045 26 EB     BNE  POS
0047 33 5F     LEAU -1,U  UN SEUL ZERO !
0049 30 04     LEAX 4,X
004B E6 C2     NEG  LDB  ,-U
004D 86 32     LDA  #$32
004F 3D        MUL
0050 108E 0064    LDY  #$64
0054 31 A6     LEAY  A,Y
0056 8D E80F    JSR  PLOT
0059 30 04     LEAX 4,X
005B 6D C4     TST  ,U
005D 26 EC     BNE  NEG
005F 3F        FIN  SWI
```



```

6D60 E6 80 INIT LDB ,X+
6D62 27 05 BEQ FINI
6D64 8D E803 JSR PUTC
6D67 20 F7 BRA INIT
6D69 39 FINI RTS

6D6A 1B 23 20 56 MESSAGE FCB $1B,$23,$20,$56,$0C
6D6E 0C
6D6F 1B 60 00 FCB $1B,$60,0

6D72 00 18 2A 42 TABLE FCB 0,$18,$2A,$42,$56,$6C,$80
6D76 56 6C 80
6D79 92 A4 B4 C4 FCB $92,$A4,$B4,$C4,$D2,$DE
6D7D D2 DE
6D7F EA F2 F6 FC FCB $EA,$F2,$F6,$FC,$FF,$FF
6D83 FF FF
6D85 FF FF FC F6 FCB $FF,$FF,$FC,$F6,$F2,$EA
6D89 F2 EA
6D8B DE D2 C4 B4 FCB $DE,$D2,$C4,$B4,$A4,$92
6D8F A4 92
6D91 80 6C 56 42 FCB $80,$6C,$56,$42,$2A,$1B,0
6D95 2A 18 00

        6D00 END DESSIN

```

On remarquera :

- le tracé par points, plus « joli » que le tracé continu,
- l'existence d'une horizontale noire (couleur : 0).

Vous pouvez essayer d'affiner ce tracé en augmentant, évidemment, le nombre de points !

32

GRAPHISME MATHÉMATIQUE CERCLE

BUT : *Gestion d'une table.*

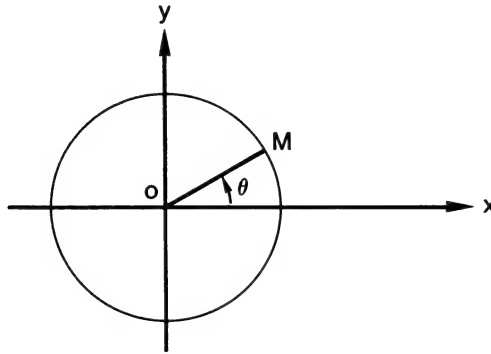
Problème : Comment tracer un cercle ?

En coordonnées cartésiennes (x, y) , le cercle centré en $0,0$ de rayon R , a pour équation :

$$x^2 + y^2 = R^2$$

Comment le faire tracer à notre ordinateur ?

Le plus simple est de prendre un « paramètre » qui est l'angle θ que fait un rayon avec l'axe x .



Ce qui donne, pour un cercle centré en $0,0$:

$$\begin{aligned}x &= R * \cos \theta \\y &= R * \sin \theta\end{aligned}$$

et pour un cercle centré en x_0, y_0 :

$$\begin{aligned}x &= R * \cos \theta + x_0 \\y &= R * \sin \theta + y_0\end{aligned}$$

Le problème est donc de déterminer le cosinus et le sinus de θ .

Comme pour le programme précédent, nous utiliserons le principe de la table.

Compte tenu des relations entre sinus et cosinus, une table suffit, si nous la limitons à 90 degrés. En effet, le sinus de l'angle θ est égal au cosinus de l'angle $(90 - \theta)$.

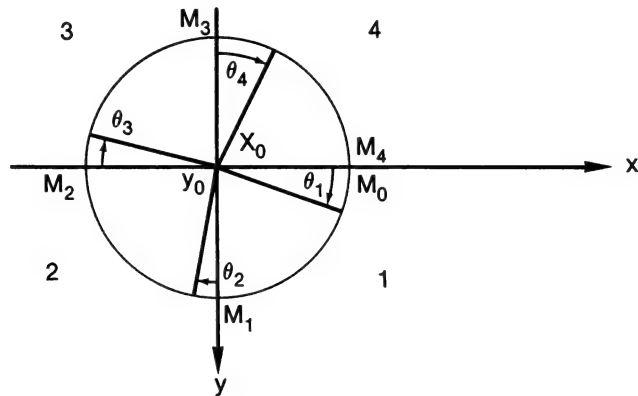
Une lecture « descendante » de la table donne les sinus, une lecture « montante » donne le cosinus selon le schéma suivant :

	sinus de		cosinus de	
	0	00	90	
	5	.18	85	
	10	.2A	80	
	.	.	.	
	80	.FC	10	
	85	.FF	5	
	90	.FF	0	
sens de lecture des sinus	↓			↑ sens de lecture des cosinus

Nous aurons donc besoin, pour un angle donné, de deux pointeurs. Dans le programme qui suit, X pointe les cosinus et sera décrémenté à chaque opération, alors que Y, qui pointe les sinus, sera incrémenté.

Compte tenu des coordonnées de l'écran avec 0,0 en haut à gauche, nous nous proposons de tracer le cercle de rayon 48 (30H) centré au milieu de l'écran, soit en $x_0 = 160$ (A0H) et $y_0 = 100$ (64H).

En utilisant les formules écrites plus haut et une table limitée à 90 degrés, nous ne pouvons tracer que le quart de cercle en bas à droite. Pour tracer les autres quarts, il faut adopter une paire de formules par quadrant et changer d'angle de référence selon le schéma ci-dessous :



Les formules à adopter et les modes opératoires sont :

QUADRANT 1

$$\begin{aligned} x &= R * \cos \theta_1 + x_0 \\ y &= R * \sin \theta_1 + y_0 \end{aligned}$$

(Y) est incrémenté à chaque calcul et pointe le *bas* de la table à la fin du tracer — Point M1 alors que

(X) qui a été décrémenté, pointe le *haut* de la table.

QUADRANT 2

Il faudra donc adopter les formules :

$$\begin{aligned} x &= -R * \sin \theta_2 + x_0 \\ y &= R * \cos \theta_2 + y_0 \end{aligned}$$

avec (X) pointant les sinus et (Y) les cosinus. En M2 (X) est en bas de la table et (Y) en haut.

QUADRANT 3

$$\begin{aligned} x &= -R * \cos \theta_3 + x_0 \\ y &= -R * \sin \theta_3 + y_0 \end{aligned}$$

QUADRANT 4

$$\begin{aligned} x &= R * \sin \theta_4 + x_0 \\ y &= -R * \cos \theta_4 + y_0 \end{aligned}$$

Le programme qui suit ne traite que le premier quadrant, essayez de le compléter pour obtenir un cercle complet en prenant garde aux raccords.
Pour améliorer le dessin, on peut augmenter le nombre de points. En travaillant de 2 en 2 degrés, il faut 46 valeurs.

```
#ESSAI GRAPHISME MATHEMATIQUE
*          CERCLE

        6038  FORME EQU      $6038
        6041  CHDRAW EQU     $6041
        E80C  DRAW EQU      $E80C
        E803  PUTC EQU      $E803
        E80F  PLOT EQU      $E80F

6D00 10CE 8000  DESSIN LDS    $$8000
6D04 8E    6D6A      LDX     #MESSAGE
6D07 BD    6D60      JSR     INIT
6D0A 7F    6041      CLR     CHDRAW  GRAPHISME
6D0D 86    05        LDA     #5      FORME
6D0F B7    6038      STA     FORME
6D12 CC    A064      LDD     #$A064  CENTRE
6D15 FD    7000      STD     $7000
6D18 86    30        LDA     #$30    RAYON
6D1A B7    7002      STA     $7002
6D1D 8E    0000      LDX     #$00
6D20 108E 0064      LDY     #$64
6D24 BD    E80F      JSR     PLOT
6D27 108E 6D72      LDY     #TABLE  SINUS
6D2B 8E    6D85      LDX     #TABLE+$13 COSINUS
6D2E A6    82        ROND  LDA     , -X
6D30 27    2D        BEQ     FIN
6D32 F6    7002      LDB     $7002
6D35 3D                MUL          (D)=R*COS
6D36 34    10        PSHS    X
6D38 1F    89        TFR     A,B
6D3A 4F                CLRA
6D3B 1F    01        TFR     D,X      (X)=R*COS ENTIER
6D3D F6    7000      LDB     $7000
6D40 4F                CLRA
6D41 30    8B        LEAX    D,X      (X)=R*COS + x0
6D43 A6    A0        LDA     ,Y+
6D45 F6    7002      LDB     $7002
6D48 3D                MUL
6D49 1F    89        TFR     A,B      (D)=R*SIN
```

```

604B 4F          CLRA
604C 34 20      PSHS  Y
604E 1F 02      TFR   D,Y
6050 F6 7001    LDB   $7001
6053 4F          CLRA
6054 31 AB      LEAY  D,Y      (Y)=R*SIN + y.
6056 BD E80F    JSR   PLOT
6059 35 20      PULS  Y
605B 35 10      PULS  X
605D 20 CF      BRA   ROND
605F 3F          FIN  SWI

6060 E6 00      INIT  LDB   ,X+
6062 27 05      BEQ   FINI
6064 BD E803    JSR   PUTC
6067 20 F7      BRA   INIT
6069 39          FINI  RTS

606A 1B 23 20 56 MESSAGE FCB  $1B,$23,$20,$56,$00
606E 0C
606F 1B 60 00          FCB  $1B,$60,0
6072 00 18 2A 42 TABLE FCB  0,$18,$2A,$42,$56,$6C,$80
6076 56 6C 80
6079 92 A4 B4 C4          FCB  $92,$A4,$B4,$C4,$D2,$DE
607D 02 DE
607F EA F2 F6 FC          FCB  $EA,$F2,$F6,$FC,$FF,$FF
6083 FF FF
        6000          END  DESSIN

```

33

JEU DU SOLITAIRE

BUT : *Apprendre en jouant...*

Le jeu du solitaire est un ensemble de pions, disposés en croix, selon le schéma ci-dessous :

```

      *  *  *
      *  *  *
    *  *  *  *  *  *  *
    *  *  *      *  *  *
    *  *  *  *  *  *  *
      *  *  *
      *  *  *
```

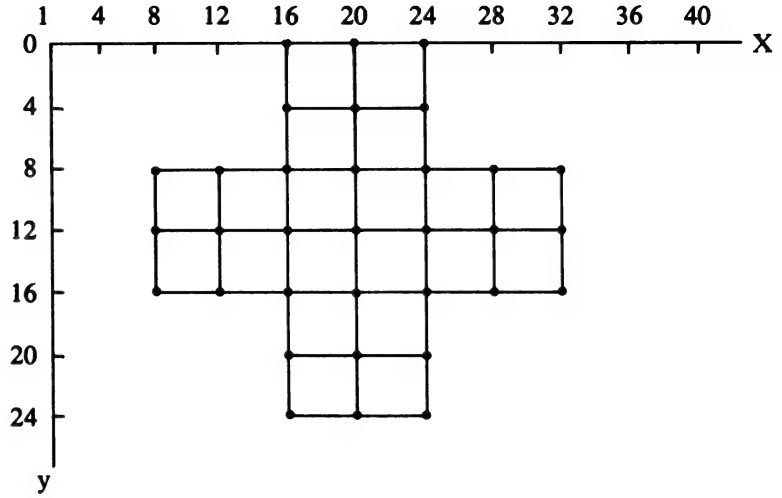
Il s'agit, en jouant « aux dames » avec les pions — on met un pion dans une place vide et on enlève le pion « sauté » ; les déplacements sont horizontaux ou verticaux — de n'en garder qu'un... **au centre !**

Il faut donc, tout d'abord, dessiner les pions, puis envisager comment les prendre et les placer.

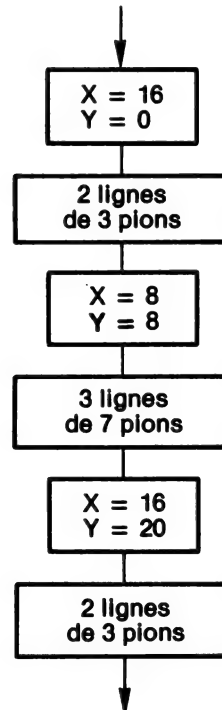
Nous disposons d'un photostyle, un pion sera un pavé (7FH en ASCII) lumineux (clair sur fond noir) et l'absence de pion, un astérisque (* codé 2A). Le crayon servira à prendre et à poser les pions. Ce qui, nous le verrons, pose un problème de coordonnées.

Il faut donc **poser** les pions, en utilisant le principe de positionnement du curseur.

On utilise **tout** l'écran ($1 \leq X \leq 40$, $0 \leq Y \leq 24$). Pour cela, il faut agrandir la fenêtre, limitée par le moniteur, le haut à la ligne 0 et le bas la ligne 24.



On arrive à l'organigramme partiel suivant :



Ce travail est réalisé en chargeant une table avec 1F (message de début de travail), Y, X, le dessin du pion et 00 (fin de message). On utilise 2 registres pour stocker le nombre de lignes et le nombre de pions. On passe d'une colonne à l'autre en augmentant l'abscisse de 4 et d'une rangée à l'autre, en augmentant l'ordonnée de 4 (sous-programmes DAMIER et DESSIN).

* JEU DU SOLITAIRE
* AVEC PHOTOSTYLE

	E803	PUTC	EQU	\$E803	
	E818	GETL	EQU	\$E818	
	E818	LPIN	EQU	\$E818	
	E824	GETS	EQU	\$E824	
7300	10CE	8000	DEBUT	LDS	#\$8000
7304	CE	7429		LDU	#INIT
7307	BD	73F6		JSR	MESS
730A	CE	7438		LDU	#TAMPON
730D	8E	1F40		LDX	#\$1F40
7310	AF	C4		STX	,U
7312	8E	7F00		LDX	#\$7F00
7315	AF	43		STX	3,U
7317	8E	0302		LDX	#\$0302
731A	86	50		LDA	\$50
731C	BD	7404		JSR	DAMIER
731F	8E	0703		LDX	#\$0703
7322	86	48		LDA	\$48
7324	BD	7404		JSR	DAMIER
7327	8E	0302		LDX	#\$0302
732A	86	50		LDA	\$50
732C	BD	7404		JSR	DAMIER

NB. PAVE NB. LIGNE
INIT X

* DEBUT DU JEU

732F	BD	7394	NON	JSR	LPEN	
7332	BD	73DA		JSR	CENTRE	
7335	26	F8		BNE	NON	
7337	BD	73AA	JEU	JSR	CRAYON	
733A	C1	7F		CMPB	#\$7F	PAVE ?
733C	26	F9		BNE	JEU	
733E	AE	41		LDX	1,U	X0 ET Y0
7340	AF	45		STX	5,U	
7342	BD	73D2		JSR	ETOILE	
7345	BD	73AA	PLACE	JSR	CRAYON	
7348	C1	2A		CMPB	#\$2A	ETOILE ?
734A	26	F9		BNE	PLACE	
734C	A6	45	JOUE	LDA	5,U	(A) = Y0
734E	A0	41		SUBA	1,U	Y0 - Y1
7350	27	12		BEQ	DELTAX	
7352	81	08		CMPA	#8	
7354	27	06		BEQ	DELTAX0	
7356	81	F8		CMPA	#-8	
7358	27	02		BEQ	DELTAX0	
735A	20	E9		BRA	PLACE	
735C	A6	46	DELTAX0	LDA	6,U	(A) = X0
735E	A1	42		CMPA	2,U	X0 = X1
7360	27	17		BEQ	ENLEVY	
7362	20	E1		BRA	PLACE	
7364	A6	46	DELTAX	LDA	6,U	
7366	A0	42		SUBA	2,U	ICI Y0 = Y1
7368	26	05		BNE	SUITE	
736A	BD	73D8	CORIGE	JSR	PAVE	
736D	20	C8		BRA	JEU	

736F	81	08	SUITE	CMPA	#8	
7371	27	15		BEQ	ENLEVX	
7373	81	F8		CMPA	#-8	
7375	27	11		BEQ	ENLEVX	
7377	20	0C		BRA	PLACE	
7379	8D	73D8	ENLEVY	JSR	PAVE	
737C	A6	45		LDA	5,U	
737E	AB	41		ADDA	1,U	
7380	44			LSRA		
7381	A7	41		STA	1,U	
7383	8D	73D2	ENLEV	JSR	ETOILE	
7386	20	AF		BRA	JEU	
7388	8D	73D8	ENLEVX	JSR	PAVE	
738B	A6	46		LDA	6,U	
738D	AB	42		ADDA	2,U	
738F	44			LSRA		
7390	A7	42		STA	2,U	
7392	20	EF		BRA	ENLEV	
7394	8D	E81B	LPEN	JSR	LPIN	
7397	24	FB		BCC	LPEN	
7399	8D	E81B		JSR	GETL	
739C	25	F6		BOS	LPEN	
739E	C6	07		LDB	##07	BEL
73A0	8D	E803		JSR	PUTC	
73A3	8D	73DC		JSR	IKS	
73A6	8D	73EA		JSR	IGREC	
73A9	39			RTS		
73AA	108E	0000	CRAYON	LDY	#0	
73AE	31	3F	TEMPO	LEAY	-1,Y	TEMPORISATION
73B0	26	FC		BNE	TEMPO	
73B2	8D	7394		JSR	LPEN	
73B5	4F			CLRA		
73B6	E6	42		LDB	2,U	
73B8	C0	40		SUBB	##40	
73BA	C1	02		CMPB	#2	
73BC	1027	FF40		LBEO	DEBUT	
73C0	1F	01		TFR	D,X	COORDONNEE X
73C2	A6	41		LDA	1,U	
73C4	80	40		SUBA	##40	COORDONNEE Y
73C6	8D	E824		JSR	GETS	
73C9	39			RTS		
73CA	AE	41	CENTRE	LDX	1,U	
73CC	8C	4C54		CMPX	##4C54	
73CF	27	01		BEQ	ETOILE	
73D1	39			RTS		
73D2	86	2A	ETOILE	LDA	##2A	
73D4	A7	43	ETOIL	STA	3,U	
73D6	20	1E		BRA	MESS	
73D8	86	7F	PAVE	LDA	##7F	
73DA	20	F8		BRA	ETOIL	
73DC	1F	10	IKS	TFR	X,D	
73DE	C4	F0		ANDB	##F0	CALCUL N2
73E0	CB	10		ADDB	##10	
73E2	56			RORB		

```

73E3 56          RORB
73E4 56          RORB
73E5 08 40       ADDB  #40
73E7 E7 42       STB  2,U
73E9 39          RTS

73EA 1F 20       IGREC TFR  Y,D
73EC 04 F0       ANDB  #F0
73EE 54          LSRB
73EF 54          LSRB
73F0 54          LSRB
73F1 08 40       ADDB  #40  CALCUL N1
73F3 E7 41       STB  1,U
73F5 39          RTS

73F6 34 40       MESS  PSHS  U
73F8 E6 00       MESS0 LDB   ,U+
73FA 27 05       BEQ   FIN
73FC 80 E803     JSR   PUTC
73FF 20 F7       BRA   MESS0
7401 35 40       FIN   PULS  U
7403 39          RTS

7404 AF 45       DAMIER STX  5,U
7406 34 02       DAME  PSHS  A
7408 A7 42       STA   2,U
740A A6 45       LDA   5,U  NB. PAVE
740C 8D 741C     JSR   DESSIN
740F A6 41       LDA   1,U
7411 8B 04       ADDA  #4    Y = Y + 4
7413 A7 41       STA   1,U
7415 35 02       PULS  A
7417 6A 46       DEC   6,U  NB. LIGNE - 1
7419 26 EB       BNE   DAME
741B 39          RTS

741C 8D 73F6     DESSIN JSR   MESS
741F E6 42       LDB   2,U
7421 08 04       ADDB  #4    X = X + 4
7423 E7 42       STB   2,U
7425 4A          DECA
7426 26 F4       BNE   DESSIN  NB. PAVE - 1
7428 39          RTS

7429 1F 20 20     INIT  FCB   $1F,$20,$20  *HAUT FENETRE
742C 1F 12 14     FCB   $1F,$12,$14  *BAS FENETRE
742F 0C 1B 47     FCB   $0C,$1B,$47  *EFFACE ET
7432 1B 50 1B 61  FCB   $1B,$50,$1B,$61  *COULEUR
7436 7F 00       FCB   $7F,00  *PAVE

7438 00          TAMPON FCB   0

              7300          END  DEBUT

```

Pour commencer, nous retirons le pion central en le pointant à l'aide du crayon optique, ce qui permet, en outre, le réglage de la luminosité. Nous sommes avertis d'une « bonne » lecture par un BIP (voir sous-programme LPEN) et nous testons la position pointée à l'aide de GETL.

Comme nous l'avons vu dans le programme 30, le programme GETL retourne la position du point visé dans X et Y avec :


$$0 \leq (X) \leq 319 \text{ et } 0 \leq (Y) \leq 199.$$

Pour lire l'état d'une case, il faut $0 \leq (X) \leq 40$ et $1 \leq (A) \leq 24$.
Il faut passer des coordonnées d'un point à celles du pion.

Le traitement est différent pour X et Y :


- pour Y il faut diviser par 8 après avoir mis à 0 le quartet de poids faible ($25 * 8 = 200$) et ajouter 40H pour le codage (sous-programme IGREC)
- pour X il faut aussi diviser par 8 mais après avoir mis le quartet de poids faible à 0 et ajouter 1 au quartet de poids fort. On ajoute 40 pour le codage (sous-programme IKS).

(Si vous n'êtes pas convaincus, faites un tableau des coordonnées des pions et des points visés à l'intérieur des pavés qui sont constitués de $8 * 8$ points).

En raison de la rapidité du 6809, il faut temporiser la lecture du crayon puis analyser « l'état » de la zone pointée : zone avec () ou sans (*) pion. On lit l'écran en faisant appel à GETS, qui met dans B le code ASCII du caractère pointé par (X) — abscisse — et (A) — ordonnée.

Attention ici : $1 \leq (X) \leq 40$ et $0 \leq (A) \leq 24$; à l'écriture il faut $41H \leq (X) \leq 78H$ et $40H \leq (Y) \leq 58H$.

Si la réponse est « sans pion », il faut que cette place soit sur la même ligne (horizontale ou verticale) que le pion précédemment pointé et à une distance égale à 8, sinon il y a erreur.

Dans le cas où les conditions sont remplies, on place le pion () et on enlève le pion intermédiaire (le pion initial étant enlevé dès sa « prise » peut être remis en place si on pointe son emplacement).

En pointant le pavé en haut, à gauche, on relance le jeu.

A vous de jouer... une solution est donnée à la fin du livre.

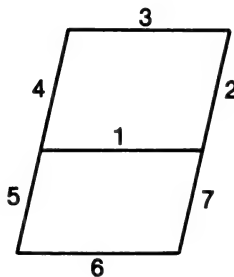
34

UN NOUVEL AFFICHAGE

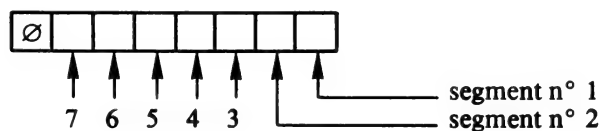
BUT : *Créer une routine qui permette d'afficher des caractères 7 segments de grande dimension.*

Utilisation de la routine DRAW.

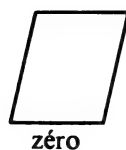
L'affichage 7 segments, comme son nom l'indique, est constitué de 7 segments que nous noterons de 1 à 7 :



A chaque segment, nous associerons un bit dans un registre de 8 bits.



On voit donc ainsi qu'à chaque chiffre sera associé une valeur. Par exemple pour :



0	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---

Soit : \$7E

La table des caractères 7 segments est pointée par GENCAR. DEPLRE permet de définir un déplacement relatif entre la position précédente et la suivante. Si le bit correspondant au segment est à un, on tracera un trait par DRAW entre les deux positions. Si le bit est à zéro, on recopiera simplement la position suivante dans les registres PLOTX et PLOTY.

L'effet d'épaisseur est obtenu par décalage de trois tracés successifs.

* BG **** UN NOUVEL AFFICHAGE **** RW *									

	E803	PUTC	EQU	\$E803	Affichage				
	E80C	DRAW	EQU	\$E80C	Trait				
	603D	PLOTX	EQU	\$603D				Recopie le dernier X	
	603F	PLOTY	EQU	\$603F				Recopie le dernier Y	
7000 30 7E 31 42	GENCAR	FCB		'0,\$7E,'1,\$42,'2,\$EC,'3	← Générateur de caractères				
7004 32 EC 33									
7007 E6 34 D2 35		FCB		\$E6,'4,\$D2,'5,\$B6,'6,\$BE					
700B B6 36 BE									
700E 37 62 38 FE		FCB		'7,\$62,'8,\$FE,'9,\$F6					
7012 39 F6									
7014 10 00 08 F0	DEPLRE	FCB		\$10,\$00,\$08,-\$10,-\$10,\$00	← Déplacements relatifs				
7018 F0 00									
701B F8 10 F8 10		FCB		-\$08,\$10,-\$08,\$10,\$10,\$00					
701E 10 00									
7020 08 F0 10 00		FCB		\$08,-\$10,\$10,\$00					
	6038	FORME	EQU	\$6038	← Registre de couleurs en mode graphique				
	0000	NOIR	EQU	\$00	Couleurs d'encore				
	0001	ROUGE	EQU	\$01					
	0002	VERT	EQU	\$02					
	0003	JAUNE	EQU	\$03					
	0004	BLEU	EQU	\$04					
	0005	MAGEN	EQU	\$05					
	0006	CYAN	EQU	\$06					
	0007	BLANC	EQU	\$07					
7024 0E 0000	DEP	LDU		#ENDMEM					
7027 8E 0040		LDX		#\$40	← Positions du premier caractère				
702A 108E 0060		LDY		#\$60					
702E 86 04		LDA		#BLEU	couleur				
7030 06 37		LDB		#17	Code ASCII				
7032 BD 7059		JSR		NQUAFF					
7035 86 07		LDA		#BLANC					
7037 06 36		LDB		#16					
7039 BD 7059		JSR		NQUAFF					
703C 86 01		LDA		#ROUGE					
703E 06 35		LDB		#15					
7040 BD 7059		JSR		NQUAFF					
7043 86 03		LDA		#JAUNE					
7045 06 38		LDB		#18					
7047 BD 7059		JSR		NQUAFF					
704A 86 02		LDA		#VERT					
704C 06 39		LDB		#19					
704E BD 7059		JSR		NQUAFF					
7051 86 06		LDA		#CYAN					
7053 06 34		LDB		#14					
7055 BD 7059		JSR		NQUAFF					
7058 3F		SWI							

7059	36	02	NOUAFF	PSHU	A	←	Sauvegarde la couleur et la position
705B	36	30		PSHU	X,Y		
705D	B7	6038		STA	FORME	←	Definie la couleur
7060	8E	7000		LDX	#GENCAR	←	Pointe sur le générateur de caractères
7063	8C	7014	SUIT	CMPX	#GENCAR+\$14	←	Fin?
7066	24	23		BHS	FIN	←	Oui.
7068	E1	81		CMPB	,X++	←	Non. Le caractère est-il égal au caractère pointé ?
706A	26	F7		BNE	SUIT	←	Non.
706C	30	1F		LEAX	-1,X	←	Oui. Alors on décrémente le pointeur pour pointer sur l'octet spécifiant le caractère 7 segments
706E	E6	84		LDB	,X		
7070	37	30		PULU	X,Y	←	On rétablit l'ordre de rangement dans la pile
7072	36	34		PSHU	X,Y,B		
7074	31	3F		LEAY	-1,Y	←	Point de départ du premier tracé
7076	30	01		LEAX	1,X		
7078	8D	708E		JSR	AFFI7	←	On trace
707B	37	34		PULU	X,Y,B	←	Pour recharger X, Y, B sans modifier le pointeur de pile
707D	36	34		PSHU	X,Y,B		
707F	31	21		LEAY	1,Y	←	Points de départ du second tracé
7081	30	1F		LEAX	-1,X		
7083	8D	708E		JSR	AFFI7	←	On trace
7086	37	34		PULU	X,Y,B	←	Points de départ du troisième tracé
7088	8D	708E		JSR	AFFI7	←	
708B	37	02	FIN	PULU	A	←	Restitue la couleur
708D	39			RTS			
708E	34	40	AFFI7	PSHS	U	←	Sauve le Pointeur
7090	BF	603D		STX	PLOTX	←	Charge les registres avec les points de départ
7093	10BF	603F		STY	PLOTY	←	
7097	0E	7014		LDU	#DEPLRE	←	Pointeur de déplacement relatif
709A	A6	00	CONT	LDA	,U+	←	Octet dans A
709C	30	86		LEAX	A,X	←	Points d'arrivée
709E	A6	00		LDA	,U+	←	
70A0	31	A6		LEAY	A,Y	←	
70A2	58			LSLB		←	Si le bit est à 1 on trace un trait
70A3	25	09		BCS	TRAI	←	
70A5	BF	603D		STX	PLOTX	←	Si le bit est à 0 on recopie les points d'arrivée dans les registres
70A8	10BF	603F		STY	PLOTY	←	
70AC	20	03		BRA	NSEG	←	Nouveau segment
70AE	8D	E00C	TRAI	JSR	DRAW	←	Un trait
70B1	1183	7024	NSEG	CMPU	#DEPLRE+16T	←	Fin ?
70B5	25	E3		BLO	CONT	←	Non. On continue
70B7	59			ROLB			
70B8	35	40		PULS	U	←	Restaure la Pile
70BA	39			RTS			
		0000		END			

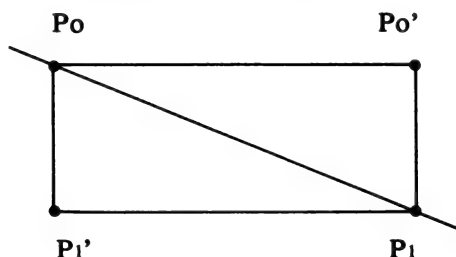
35

RECTANGLE VIDE EN MODE GRAPHIQUE

BUT : *Réaliser une routine qui permette de tracer un rectangle vide.*

Nombreuses sont les applications, à base de constructions graphiques, qui nécessitent des formes rectangulaires. Il est donc utile de disposer d'une routine qui permette de construire ces « *rectangles vides* ».

Un rectangle est parfaitement défini sur un plan par les coordonnées des deux angles situés sur la même diagonale. Nous désignerons ces points par P_0 et P_1 .



Si on affecte à P_0 les coordonnées X_0 et Y_1
et à P_1 les coordonnées X_1 et Y_0

On peut en déduire les coordonnées de P_0' et P_1'

d'où : $P_0 (X_0, Y_0)$ $P_0' (X_1, Y_0)$
 $P_1' (X_0, Y_1)$ $P_1 (X_1, Y_1)$

RVIPO est la routine qui exécute ce tracé. Les paramètres X_0, Y_0, X_1, Y_1 sont passés par l'intermédiaire de 8 octets réservés et pointés par DIAGO.

Le registre CHDRAW précise le mode en cours (00 graphique, 01 caractère).

Le registre FORME spécifie la couleur d'« encre » ou de « papier » selon la règle suivante :

(FORME) > 0 → encre (FORME) < 0 → papier

Les coordonnées des points P_0, P_0', P_1, P_1' sont chargées à tour de rôle dans les registres Y et X avant d'appeler la routine DRAW qui trace un trait entre le dernier point du trait précédent et le point pointé par les registres X et Y.

* BG *****RECTANGLE VIDE***** RW *

* DESSINE EN MODE TRAIT UN RECTANGLE *
* VIDE DONT LES COORDONNEES SONT : *
* X0,Y0-X1,Y0-X1,Y1-X0,Y1 *
* LES VALEURS X0,Y0,X1,Y1 SONT PASSEES *
* PAR LES REGISTRES POINTES PAR DIAGO *
* DIAGO --> X0 DIAGO+2 --> Y0 *
* DIAGO+4 --> X1 DIAGO+6 --> Y1 *
* LES COULEURS SONT DEFINIES PAR LE *
* CONTENU DU REGISTRE FORME *
* *

		E80F	PLOT	EQU	\$E80F	Point	
		E80C	DRAW	EQU	\$E80C	Trait	
		6041	CHDRAW	EQU	\$6041		Mode point ou caractère
7000				ORG	\$7000		Origine du programme objet
7000			DIAGO	RMB	8		Réserve une zone pour définir les coordonnées d'une diagonale
		6038	FORME	EQU	\$6038		Registre qui spécifie les couleurs de papier ou d'encre
7008	CE	C000	DEBUT	LDU	#ENDMEM		Pointeur de la pile utilisateur
7008	8E	0050		LDX	#10T*8		Changement des coordonnées X0, Y0, X1, Y1 en mode graphique
700E	BF	7000		STX	DIAGO		
7011	8E	00A0		LDX	#20T*8		
7014	BF	7002		STX	DIAGO+2		
7017	8E	00F0		LDX	#30T*8		
701A	BF	7004		STX	DIAGO+4		
701D	8E	0028		LDX	#05T*8		
7020	BF	7006		STX	DIAGO+6		
7023	86	03		LDA	#\$03		Couleur de papier : jaune
7025	B7	6038		STA	FORME		
7028	86	00		LDA	#\$00		Mise en mode graphique
702A	B7	6041		STA	CHDRAW		
702D	BD	7031		JSR	RVIPO		
7030	3F			SWI			
7031	36	36	RVIPO	PSHU	X,Y,A,B		Sauvegarde l'état des registres internes du 6809
7033	10BE	7002		LDY	DIAGO+2	On charge Y0	
7037	BE	7000		LDX	DIAGO	Puis X0	Ecrit un point de coordonnées Y0, X0. Les registres PLOTX et PLOTY sont chargés avec X0 et Y0
703A	BD	E80F		JSR	PLOT		
703D	BE	7004		LDX	DIAGO+4	Charge X1	
7040	BD	E80C		JSR	DRAW		Trace un trait entre X0, Y0 et X1, Y0
7043	10BE	7006		LDY	DIAGO+6	Charge Y1	
7047	BD	E80C		JSR	DRAW		Trace un trait entre X1, Y0 et X1, Y1
704A	BE	7000		LDX	DIAGO	Charge X0	
704D	BD	E80C		JSR	DRAW		Trace un trait entre X1, Y1 et X0, Y1
7050	10BE	7002		LDY	DIAGO+2	Charge Y0	
7054	BD	E80C		JSR	DRAW		Trace un trait entre X0, Y1 et X0, Y0
7057	39			RTS			
		0000		END			

36

RECTANGLE VIDE EN MODE CARACTÈRE

BUT : *Réaliser une routine qui permette de tracer un rectangle vide en mode caractère.*

A partir du programme précédent, il est assez facile de construire le programme que nous vous proposons.

RVICA est en tous points semblable à RVIPO. Seuls les paramètres d'entrée diffèrent.

Le paramètre de couleur est passé par COLOR au lieu de FORME et spécifie la couleur de papier et d'encre.

CHDRAW contient le code ASCII du caractère utilisé pour le tracé.

Les coordonnées Xo, Yo, X1, Y1 pointées par DIAGO doivent être exprimées en valeurs caractères.

Notons qu'il faut **huit × huit = 64** points graphiques pour **un** point caractère.

```
*****
* BG *****RECTANGLE VIDE***** RW *
*****
```

```
*****
* DESSINE EN MODE CARACTERE UN RECTAN *
* GLE VIDE DONT LES COORDONNEES SONT: *
*      Xo.Yo-X1.Yo-X1.Y1-Xo.Y1      *
* LES VALEURS Xo.Yo,X1,Y1 SONT PASSEES *
* PAR LES REGISTRES POINTES PAR DIAGO *
*      DIAGO --> Xo      DIAGO+2 --> Yo      *
*      DIAGO+4 --> X1      DIAGO+6 --> Y1      *
* LES COULEURS SONT DEFINIES PAR LE *
*      CONTENU DU REGISTRE COLOR      *
*                                     *
*****
```

```
E80F  PLOT  EQU  $E80F  Point
E80C  DRAW  EQU  $E80C  Trait
6041  CHDRAW EQU  $6041
```

7000

ORG \$7000

7000

DIAGO RMB 8

```
603B  COLOR EQU  $603B
```

Mode point ou caractère

Origine du programme objet

Réserve une zone pour définir les coordonnées d'une diagonale

Registre qui spécifie les couleurs de papier et d'encre

7008	CE	0000	DEBUT	LDU	#ENDMEM	←	Pointeur de la pile utilisateur
7008	8E	000A		LDX	#10T	←	Chargement des coordonnées caractères X0, Y0, X1, Y1
700E	BF	7000		STX	DIAGO		
7011	8E	0014		LDX	#20T		
7014	BF	7002		STX	DIAGO+2		
7017	8E	001E		LDX	#30T		
701A	BF	7004		STX	DIAGO+4		
701D	8E	0005		LDX	#05T		
7020	BF	7006		STX	DIAGO+6		
7023	86	00E		LDA	#\$0B	←	Couleur de papier : jaune et d'encre : rouge
7025	B7	603B		STA	COLOR		
7028	86	30		LDA	#\$30		Caractere "0"
702A	B7	6041		STA	CHDRAW		
702D	BD	7031		JSR	RVICA		
7030	3F			SWI			
7031	36	36	RVICA	PSHU	X,Y,A,B	←	Sauvegarde l'état des registres internes du 6809
7033	10BE	7002		LDY	DIAGO+2	On charge Y0	
7037	BE	7000		LDX	DIAGO	Puis X0	
703A	BD	E80F		JSR	PLOT	←	Ecrit un point de coordonnées Y0, X0. Les registres PLOTX et PLOTY sont chargés avec X0 et Y0
703D	BE	7004		LDX	DIAGO+4	Change X1	
7040	BD	E80C		JSR	DRAW	←	Trace un trait entre X0, Y0 et X1, Y0
7043	10BE	7006		LDY	DIAGO+6	Change Y1	
7047	BD	E80C		JSR	DRAW	←	Trace un trait entre X1, Y0 et X1, Y1
704A	BE	7000		LDX	DIAGO	Change X0	
704D	BD	E80C		JSR	DRAW	←	Trace un trait entre X1, Y1 et X0, Y1
7050	10BE	7002		LDY	DIAGO+2	Change Y0	
7054	BD	E80C		JSR	DRAW	←	Trace un trait entre X0, Y1 et X0, Y0
7057	39			RTS			
		0000		END			

37

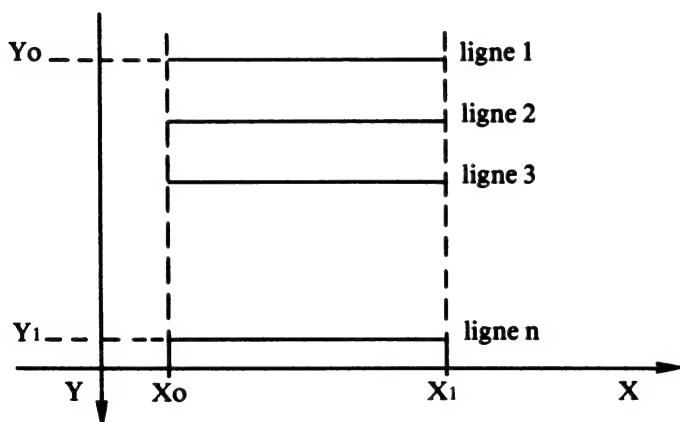
RECTANGLE PLEIN EN MODE GRAPHIQUE

BUT : Réaliser une routine qui permette de tracer un rectangle plein.

INSTRUCTION UTILISEE : EXG.

Comme pour les cas précédents, les applications graphiques nécessitent souvent de réaliser des zones pleines appelées « *Rectangles pleins* ».

La routine proposée RPLGR est déduite de la routine « rectangle vide en mode graphique » et consiste à tracer un ensemble de lignes à l'intérieur d'une zone définie par les coordonnées passées par DIAGO (X_0 , Y_0 , X_1 , Y_1).



La technique pour tracer ces lignes consiste donc, pour chaque ligne, à incrémenter la valeur de Y, jusqu'à ce que Y soit égal à Y_1 .

Une difficulté apparaît si Y_1 est plus petit que Y_0 . Dans ce cas, 2 solutions peuvent résoudre ce problème :

- 1 - Décrémenter la valeur de Y au lieu de l'incrémenter.
- 2 - Intervertir Y_0 et Y_1 afin de rétablir l'hypothèse de départ qui était :

$$Y_0 < Y_1.$$

C'est cette dernière solution que nous avons retenue dans le programme, car une instruction du 6809

EXG

permet d'échanger les contenus de registres de même capacité. Cette modification éventuelle effectuée, la suite du programme est sans difficulté.

Chaque début de ligne est positionné par l'appel de la routine PLOT, qui permet de charger respectivement les registres PLOTX et PLOTY avec les valeurs Xo et la valeur courante de Y. La fin de la ligne en cours est spécifiée par X1 chargé à partir de DIAGO + 4 et la valeur de Y.

Si Y égal à Y1, le trait précédent était le dernier.

* BG *****RECTANGLE PLEIN***** RW *
***** GRAPHIQUE *****

* DESSINE EN MODE GRAPHIQUE UN RECTAN *
* GLE PLEIN DONT LES COORDONNEES SONT: *
* Xo,Yo-X1,Yo-X1,Y1-Xo,Y1 *
* LES VALEURS Xo,Yo,X1,Y1 SONT PASSEES *
* PAR LES REGISTRES POINTES PAR DIAGO *
* DIAGO --> Xo DIAGO+2 --> Yo *
* DIAGO+4 --> X1 DIAGO+6 --> Y1 *
* LA COULEUR EST DEFINIE PAR LE *
* CONTENU DU REGISTRE FORME *
* *

E80F PLOT EQU \$E80F Point
E80C DRAW EQU \$E80C Trait
6041 CHDRAW EQU \$6041

7100 ORG \$7100

7100 DIAGO RMB 8

 6038 FORME EQU \$6038

7100 CE C000 DEBUT LDU #ENDMEM

7100 8E 0050 LDX #10T*8
710E BF 7100 STX DIAGO
7111 8E 00A0 LDX #20T*8
7114 BF 7102 STX DIAGO+2
7117 8E 00F0 LDX #30T*8
711A BF 7104 STX DIAGO+4
711D 8E 0028 LDX #05T*8
7120 BF 7106 STX DIAGO+6
7123 86 FE LDA #-\$2
7125 B7 6038 STA FORME
7128 86 00 LDA #\$00
712A B7 6041 STA CHDRAW
712D BD 7131 JSR RPLGR
7130 3F SWI
7131 36 36 RPLGR PSHU X,Y,A,B
7133 10BE 7102 LDY DIAGO+2
7137 10BC 7106 CMPY DIAGO+6
7138 23 0C BLS TRACE
713D BE 7106 LDX DIAGO+6

Mode graphique ou caractère

Origine du programme objet

Réserve une zone pour définir les coordonnées d'une diagonale

Registre qui spécifie les couleurs de papier ou d'encre

Pointeur de la pile utilisateur

Chargement des coordonnées Xo, Yo, X1, Y1

Couleur de papier : rouge

Mise en mode graphique

Sauvegarde l'état des registres internes du 6809

On charge Yo
Compare Yo et Y1

Si plus petit ou égal, on ne modifie rien
Si plus grand on échange Yo et Y1

7140	1E	12		EXG	X,Y	
7142	BF	7106		STX	DIAG0+6	
7145	10BF	7102		STY	DIAG0+2	
7149	BE	7100	TRACE	LDX	DIAG0	Puis Xo
714C	BD	E80F		JSR	PLOT	←
714F	BE	7104		LDX	DIAG0+4	Charge X1
7152	BD	E80C		JSR	DRAW	←
7155	10BC	7106		CMPLY	DIAG0+6	←
7159	27	04		BEQ	ZE	←
715B	31	21		LEAY	1,Y	←
715D	20	EA		BRA	TRACE	
715F	39		ZE	RTS		
		0000		END		

Ecrit un point de coordonnées Yo, Xo.
Les registres PLOTX et PLOTY
sont chargés avec Xo et Yo

Trace un trait entre Xo, Yn et X1, Yn

Sommes-nous à la fin du tracé ?

Oui. Alors on saute à la fin

Non. On incrémente Y de 1 pour tracer
le trait suivant

38

RECTANGLE PLEIN EN MODE CARACTÈRE

BUT : *Apprendre à transformer des coordonnées graphiques en coordonnées caractères et à utiliser le crayon optique pour pointer les points d'entrée.*

Dans les trois tracés de rectangles précédents (vides ou pleins), nous avons défini les points d'entrée dans le corps du programme. Nous vous proposons maintenant d'utiliser le crayon optique pour pointer les emplacements où l'on souhaite faire apparaître un rectangle plein, en mode caractère.

Pour éviter de relancer le programme après chaque tracé, nous le rebouclerons sur lui-même via l'étiquette **SUIT**. A chaque boucle, nous incrémenterons le registre **CHDRAW** pour modifier le caractère, afin de bien repérer les limites du dernier tracé.

Premier problème : La machine va beaucoup plus vite que l'humain. A peine aurez-vous posé le crayon sur l'écran pour pointer le premier point que déjà la machine sera au second. Résultat : les deux points seront identiques et votre rectangle sera réduit à l'aire d'un caractère. Il convient donc d'insérer une pause, au moins égale au temps de réaction humain, entre les deux pointages. Cette pause est implantée dans le sous-programme **CRAY** par la boucle **AR2**. La valeur **\$4FFF** spécifie le temps de pause (vous pouvez la modifier).

Second problème : Le crayon optique restitue dans les registres **X** et **Y** les coordonnées pointées en valeurs graphiques. Soit 0 à 319 en **X** et 0 à 199 en **Y**. Un point caractère correspond à un carré de 8 par 8 points graphiques. Le passage de coordonnées graphiques aux coordonnées caractères s'effectuera donc par une division par 8.

Comment diviser par 8 ? Nous avons vu, dans les exemples précédents (division), qu'un décalage à droite d'un nombre binaire équivaut à sa division par 2. Trois décalages à droite seront donc équivalents à une division par 8 ($2^3 = 8$). Dans le 6809, il n'existe pas d'instruction de décalage sur les registres **X** et **Y**, et de plus sur 16 bits. Il faut donc passer par le registre **D** qui est la concaténation des registres **A** et **B** que nous décalerons successivement.

Quelles instructions utiliser ? Il faut remarquer qu'après le décalage, le bit de plus faible poids du registre **A** doit devenir le bit de plus fort poids du registre **B**. Il convient donc de ne pas le perdre « en route ». Pour cela, nous utiliserons l'instruction **LSRA** qui est un décalage logique à droite et qui charge le bit **C** du carry avec le bit de poids faible de **A**. Le décalage sur **B** sera réalisé par une rotation à droite (**RORB**). Instruction qui permet de charger le bit de poids fort de **B** avec le bit de retenue **C**. Cette séquence, répétée trois fois, permettra d'obtenir la division par 8 du registre **D**.

Pourquoi incrémenter les valeurs X_0 et X_1 après la division ? Les coordonnées graphiques en X sont exprimées de 0 à 319 (320 points) tandis que les coordonnées caractères sont exprimées de 1 à 40. Il existe donc un décalage de 1 après la division. Ecart qu'il faut, bien entendu, supprimer, d'où l'incrémentation de un sur les valeurs faibles de X_0 et X_1 . Notez qu'en Y les coordonnées graphiques sont exprimées de 0 à 199 (200 points) et les coordonnées caractères de 0 à 24. Il n'y a donc pas de décalage après la division.

```
*****
* BG *****RECTANGLE PLEIN***** RW *
***** CARACTERE *****
```

```
*****
* DESSINE EN MODE CARACTERE UN RECTAN *
* GLE PLEIN DONT LES COORDONNEES SONT: *
*      X0,Y0-X1,Y0-X1,Y1-X0,Y1      *
* LES VALEURS X0,Y0,X1,Y1 SONT PASSEES *
*      PAR LE CRAYON OPTIQUE          *
* A CHAQUE RECTANGLE LE CARACTERE    *
*      EST MODIFIE                    *
* LES COULEURS SONT DEFINIES PAR LE   *
*      CONTENU DU REGISTRE COLOR      *
*                                     *
*****
```

E803	PUTC	EQU	\$E803	Ecran
E81B	LPIN	EQU	\$E81B	Contact Crayon
E818	GETL	EQU	\$E818	Crayon
E80F	PLOT	EQU	\$E80F	Point
E80C	DRAW	EQU	\$E80C	Trait
6041	CHDRAW	EQU	\$6041	

Mode graphique ou caractère

7100		DIAGO	RMB	8	← Réserve une zone pour définir les coordonnées d'une diagonale
	603B	COLOR	EQU	\$603B	← Registre qui spécifie les couleurs de papier et d'encre
7108 1B 23 20 57	ECBL	FCB		\$1B,\$23,\$20,\$57,\$00	
710C 00					

710D CE	C000	DEBUT	LDU	#ENDMEM	← Pointeur de la pile utilisateur
---------	------	-------	-----	---------	-----------------------------------

7110 86	30		LDA	#\$30	
7112 B7	6041		STA	CHDRAW	
7115 8E	7108	SUIT	LDX	#ECBL	Ecran blanc
7118 E6	80	AR1	LDB	,X+	
711A 27	05		BEQ	AV1	
711C B0	E803		JSR	PUTC	
711F 20	F7		BRA	AR1	
7121 B0	7195	AV1	JSR	CRAY	← Chargement des coordonnées X0, Y0, X1, Y1
7124 1F	10		TFR	X,D	

7126 44		LSRA		← Pour diviser par 8 X0 contenu dans le registre D
7127 56		RORB		
7128 44		LSRA		
7129 56		RORB		
712A 44		LSRA		
712B 56		RORB		

712C	FD	7100	STD	DIAGO		
712F	7C	7101	INC	DIAGO+1		
7132	1F	20	TFR	Y,D		
7134	44		LSRA		←	Pour diviser par 8 Yo contenu dans le registre D
7135	56		RORB			
7136	44		LSRA			
7137	56		RORB			
7138	44		LSRA			
7139	56		RORB			
713A	FD	7102	STD	DIAGO+2		
713D	BD	7195	JSR	CRAY		
7140	1F	10	TFR	X,D		
7142	44		LSRA		←	Pour diviser par 8 X1 contenu dans le registre D
7143	56		RORB			
7144	44		LSRA			
7145	56		RORB			
7146	44		LSRA			
7147	56		RORB			
7148	FD	7104	STD	DIAGO+4		
7148	7C	7105	INC	DIAGO+5		
714E	1F	20	TFR	Y,D		
7150	44		LSRA		←	Pour diviser par 8 Y1 contenu dans le registre D
7151	56		RORB			
7152	44		LSRA			
7153	56		RORB			
7154	44		LSRA			
7155	56		RORB			
7156	FD	7106	STD	DIAGO+6		
7159	26	0B	LDA	##0B	←	Couleur de papier : jaune et encre rouge
715B	27	603B	STA	COLOR		
715E	7C	6041	INC	CHDRAW		
7161	BD	7166	JSR	RPLCA		
7164	20	AF	BRA	SUIT		
7166	36	36	RPLCA	PSHU	X,Y,A,B	← Sauvegarde l'état des registres internes du 6809
7168	10BE	7102	LDY	DIAGO+2	On charge Yo	
716C	10BC	7106	CMPLY	DIAGO+6	Compare Yo et Y1	
7170	23	0C	BLS	TRACE	←	Si plus petit ou égal, on ne modifie rien
7172	BE	7106	LDX	DIAGO+6	←	Si plus grand, on échange Yo et Y1
7175	1E	12	EXG	X,Y		
7177	BF	7106	STX	DIAGO+6		
717A	10BF	7102	STY	DIAGO+2		
717E	BE	7100	LDX	DIAGO	Puis Xo	← Ecrit un point de coordonnées Yo, Xo. Les registres PLOTX et PLOTY sont chargés avec Xo et Yo
7181	BD	E80F	JSR	PLOT		
7184	BE	7104	LDX	DIAGO+4	Charge X1	
7187	BD	E80C	JSR	DRAW	←	Trace un trait entre Xo, Yn et X1, Yn
718A	10BC	7106	CMPLY	DIAGO+6	←	Sommes-nous à la fin du tracé ?

718E	27	04		BEQ	ZE	←	Oui. Alors on saute à la fin
7190	31	21		LEAY	1,Y	←	Non. On incrémente Y de 1 pour tracer le trait suivant
7192	20	EA		BRA	TRACE		
7194	39		ZE	RTS			
7195	8E	4FFF	CRAY	LDX	#\$4FFF	←	Crayon optique
7198	30	1F	AR2	LEAX	-1,X		Tempo
719A	26	FC		BNE	AR2		
719C	80	E818	ARA	JSR	GETL		Lecture
719F	25	FB		BCS	ARA		
71A1	80	E81B		JSR	LPIN		Validation
71A4	24	F6		BCC	ARA		
71A6	39			RTS			
		0000		END			

Nous avons « omis » la directive ORG, le programme commence à la première page (256 octets) libre.

39

POINTEUR VISIBLE

BUT : Apprendre à travailler sur les mémoires écran, tant caractère que couleur.

Vous avez sans aucun doute remarqué, en utilisant le crayon optique avec les programmes précédents, qu'il serait plus agréable de situer, sur l'écran, la zone pointée.

C'est ce que nous vous proposons avec cette application.

L'écran de votre ordinateur est constitué d'une page de 320 par 200 points. Chaque point peut prendre une couleur dite de « *papier ou fond* » ou bien d'« *encre ou forme* ». Ces couleurs sont aux nombres de 8 pour le TO7 et 16 pour le TO7-70. La mémoire qui gère ces états est dite « *mémoire caractère* ». Les couleurs, de papier ou d'encre, sont définies par une mémoire dite « *mémoire couleur* ». Ces deux blocs de mémoire (8 K octets) ont la même adresse logique vue du 6809. Le bit 0 du registre E7C3 permet de sélectionner l'accès à ces mémoires :

bit 0 de E7C3 = 1 → mémoire caractère

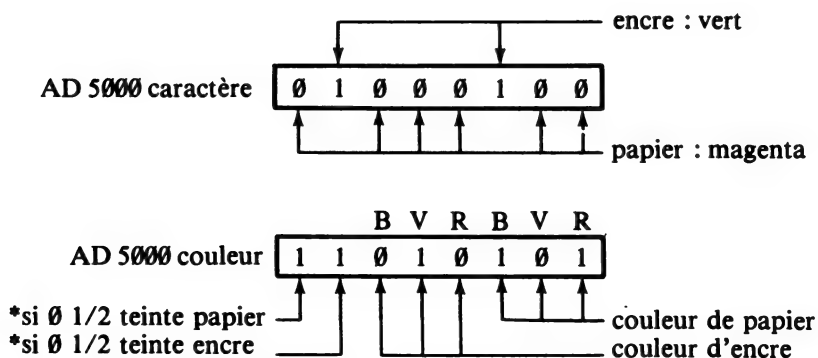
bit 0 de E7C3 = 0 → mémoire couleur

Chaque bit de la mémoire caractère correspond à un point sur l'écran :

bit à 1 → couleur d'encre

bit à 0 → couleur de papier

La mémoire couleur gère les couleurs de papier et d'encre. Chaque octet de la mémoire couleur définit la couleur des points de l'octet correspondant dans la mémoire caractère. Ainsi, par exemple, à l'adresse \$5000, on peut avoir :



*toujours à 1 sur TO7.

Code des couleurs :

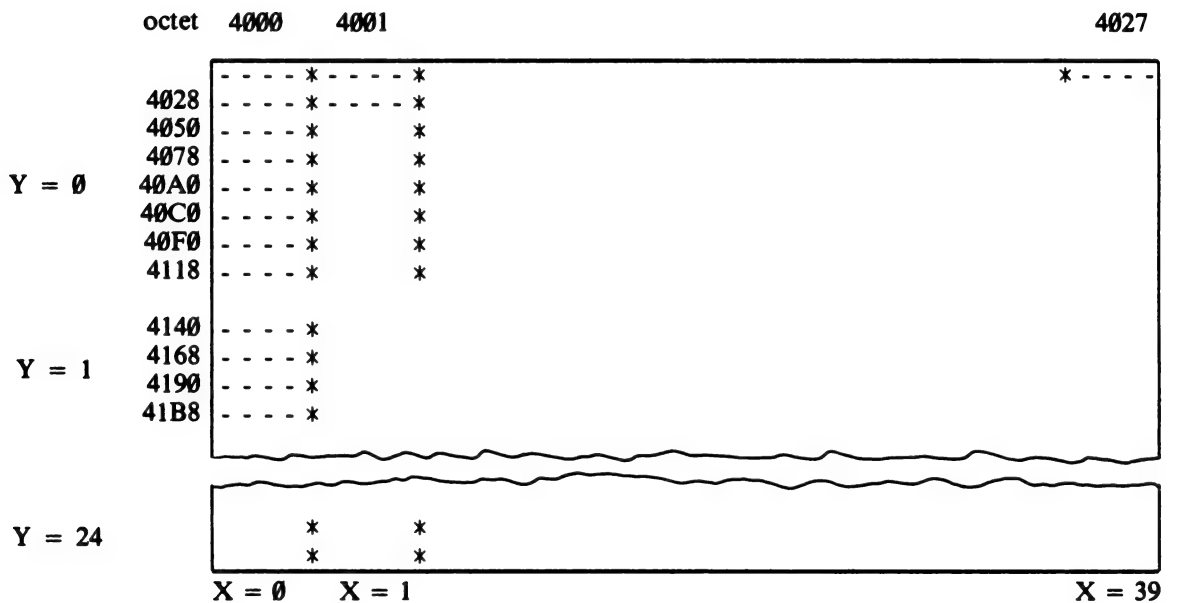
1/2	B	V	R		1/2	B	V	R	
1	0	0	0	NOIR	0	0	0	0	GRIS
1	0	0	1	ROUGE	0	0	0	1	ROSE
1	0	1	0	VERT	0	0	1	0	VERT CLAIR
1	0	1	1	JAUNE	0	0	1	1	SABLE
1	1	0	0	BLEU	0	1	0	0	BLEU CLAIR
1	1	0	1	MAGENTA	0	1	0	1	PARME
1	1	1	0	CYAN	0	1	1	0	BLEU CIEL
1	1	1	1	BLANC	0	1	1	1	ORANGE

Pour chaque octet de la mémoire caractère, il ne peut donc y avoir qu'une seule couleur d'encre et une seule couleur de papier. La modification de couleur d'un point (encre ou papier) change également les points de l'octet qui sont dans le même état.

Un point caractère (= 8 * 8 points graphiques) est parfaitement défini par 8 octets de la mémoire caractère et les 8 octets correspondants de la mémoire couleur.

La première étape du programme consiste à transformer les données graphiques du crayon optique en coordonnées caractères afin de positionner la visée sur un point caractère. Il faut ensuite lire chaque octet du point caractère afin de les sauvegarder dans la pile U.

Les mémoires écran (caractère et couleur) sont implantées sur TO7 et TO7-70 entre les adresses \$4000 et \$5FFF et organisées de la façon suivante :



L'adresse du premier octet de chaque point caractère est définie par l'équation :

$$\$4000 + 320Y + X$$

(320 = 8 lignes de 40 octets)
Y et X sont les coordonnées du point caractère.

Problème : Comment résoudre cette équation en assembleur ?

Y est contenu dans le registre D et peut être compris entre 0 et 24. Le registre A qui contient les poids forts est donc constamment à 00, et peut recevoir le multiplicateur à condition que celui-ci soit inférieur à 255 (\$FF). Horreur !! Le multiplicateur vaut 320 !

Mais 320 c'est aussi $160 * 2$. Nous multiplierons donc d'abord par 160 puis par 2. La multiplication par 2 est obtenue par décalage à gauche de B, puis de A. Le registre D contient alors le produit 320Y. L'ajout de X est réalisé par l'addition de D à X (LEAX D, X). De la même façon, l'addition de \$4000 est obtenue par LEAY ECRAN, X.

La suite du programme est sans difficulté.

- On lit et on sauve dans la pile U les octets du point caractère.
- On force le pointeur.
- On temporise pour que le pointeur soit visible.
- Puis on restitue l'état initial du point caractère.
- Si le contact du crayon optique est fermé, on sort, sinon, on recommence.

```
*****
* BG ***** POINTEUR VISIBLE ***** RW *
*****
```

```
*****
* POCRA VISUALISE LA ZONE POINTEE PAR *
* LE CRAYON OPTIQUE.                  *
* SI LE CONTACT EST FERME LES COORDON-*
* NEES SONT RETOURNEES DANS LES      *
* REGISTRES X ET Y                    *
*****
```

E803	PUTC	EQU	\$E803	Ecran
E81B	LPIN	EQU	\$E81B	Contact Crayon
E818	GETL	EQU	\$E818	Crayon
4000	ECRAN	EQU	\$4000	
E7C3	SELEC	EQU	\$E7C3	

Adresse de début de l'écran
 Registre qui permet de sélectionner
 les mémoires couleur ou forme de
 l'écran

```
7100 1B 23 20 57 ECBL FCB $1B,$23,$20,$57,$00
7104 00
```

```
7105 CE C000 DEBUT LDU #ENDMEM
7108 8E 7100 SUIT LDX #ECBL Ecran blanc
710B E6 00 AR1 LDB ,X+
710D 27 05 BEQ AV1
710F BD E803 JSR PUTC
7112 20 F7 BRA AR1
7114 BD 7118 AV1 JSR POCRA
7117 3F SWI

7118 BD 7102 POCRA JSR CRAY Crayon optique
711B BD E81B JSR LPIN Contact du lPin
711E 24 01 BCC *+3
7120 39 RTS
7121 1F 10 TFR X,D Division Par 8
7123 44 LSRA
7124 56 RORB
7125 44 LSRA
7126 56 RORB
7127 44 LSRA
7128 56 RORB
7129 1F 01 TFR D,X Division Par 8
712B 1F 20 TFR Y,D
```

712D	54		LSRB			
712E	54		LSRB			
712F	54		LSRB			
7130	86	A0	LDA	#160T	←	Pour rechercher l'adresse du premier octet du bloc caractère soit : l'adresse de début de l'écran plus 320 fois la coordonnée Y plus la coordonnée X
7132	30		MUL			
7133	58		ASLB			Multiplie Par 2
7134	49		ROLA			
7135	30	8B	LEAX	D,X	←	Ajoute la valeur de X
7137	31	89 4000	LEAY	ECRAN,X		Ajoute l'écran
713B	BD	717B	JSR	MFR	←	Mémoire forme active
713E	BD	71AE	JSR	LBLC	←	Lecture de la mémoire forme
7141	BD	716E	JSR	MCO	←	Mémoire couleur active
7144	BD	71AE	JSR	LBLC	←	Lecture de la mémoire couleur
7147	06	0F	LDB	#\$0F	←	Couleurs du pointeur
7149	BD	719C	JSR	PTER		Pointeur
714C	BD	717B	JSR	MFR	←	Mémoire forme active
714F	06	FF	LDB	#\$FF		Forme
7151	BD	719C	JSR	PTER		Pointeur
7154	8E	2000	LDX	#\$2000		Tempo
7157	30	1F	LEAX	-1,X		
7159	26	FC	BNE	*-2		
715B	BD	716E	JSR	MCO	←	Mémoire couleur active
715E	31	A9 0118	LEAY	280,Y	←	Le premier octet dépillé est le dernier empilé
7162	BD	7188	JSR	RSTI	←	Restitue le contenu de la mémoire couleur
7165	BD	717B	JSR	MFR		Mémoire forme
7168	BD	7188	JSR	RSTI	←	Restitue le contenu de la mémoire forme
716B	16	FFAA	LBRA	POCRA		
716E	36	02	PSHU	A	←	Rend la mémoire couleur active
7170	86	FE	LDA	#\$FE		
7172	B4	E7C3	ANDA	SELEC	←	Force le bit 0 du registre à 0
7175	B7	E7C3	STA	SELEC		
7178	37	02	PULU	A		
717A	39		RTS			
717B	36	02	PSHU	A	←	Rend la mémoire forme active
717D	86	01	LDA	#\$01		
717F	BA	E7C3	ORA	SELEC	←	Force le bit 0 du registre à 1
7182	B7	E7C3	STA	SELEC		
7185	37	02	PULU	A		
7187	39		RTS			
7188	34	26	RSTI	PSHS	A,B,Y	← Restitue le bloc caractère
718A	86	08	LDA	#08		8 octets
718C	31	A8 28	LEAY	40T,Y		
718F	31	A8 D8	LEAY	-40T,Y	←	Chaque octet du bloc est séparé par 40 adresses
7192	37	04	PULU	B		
7194	E7	A4	STB	,Y		
7196	4A		DECA			Compte les octets
7197	26	F6	BNE	*-8		
7199	35	26	PULS	A,B,Y		
719B	39		RTS			

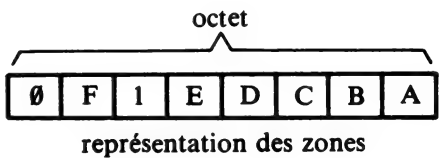
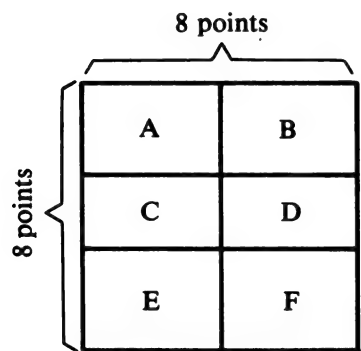
719C	34	26	PTER	PSHS	A,B,Y	← Définit la couleur ou la forme du pointeur
719E	86	08		LDA	#08	Toujours 8 octets
71A0	31	A8 D8		LEAY	-40T,Y	← 40 adresses entre chaque octet du bloc
71A3	31	A8 28		LEAY	40T,Y	
71A6	E7	A4		STB	,Y	
71A8	4A			DECA		Compte les octets
71A9	26	F8		BNE	*-6	
71AB	35	26		PULS	A,B,Y	
71AD	39			RTS		
71AE	34	26	LBLC	PSHS	A,B,Y	
71B0	86	08		LDA	#08	Encore 8 octets
71B2	31	A8 D8		LEAY	-40T,Y	
71B5	31	A8 28		LEAY	40T,Y	← 40 adresses entre chaque octet du bloc
71B8	E6	A4		LDB	,Y	
71BA	36	04		PSHU	B	
71BC	4A			DECA		Compteur
71BD	26	F6		BNE	*-8	
71BF	35	26		PULS	A,B,Y	
71C1	39			RTS		
71C2	BD	E818	CRAY	JSR	GETL	← Lecture du crayon optique
71C5	25	FB		BCS	*-3	
71C7	39			RTS		
		0000		END		

40

CARACTÈRES TÉLÉTEL

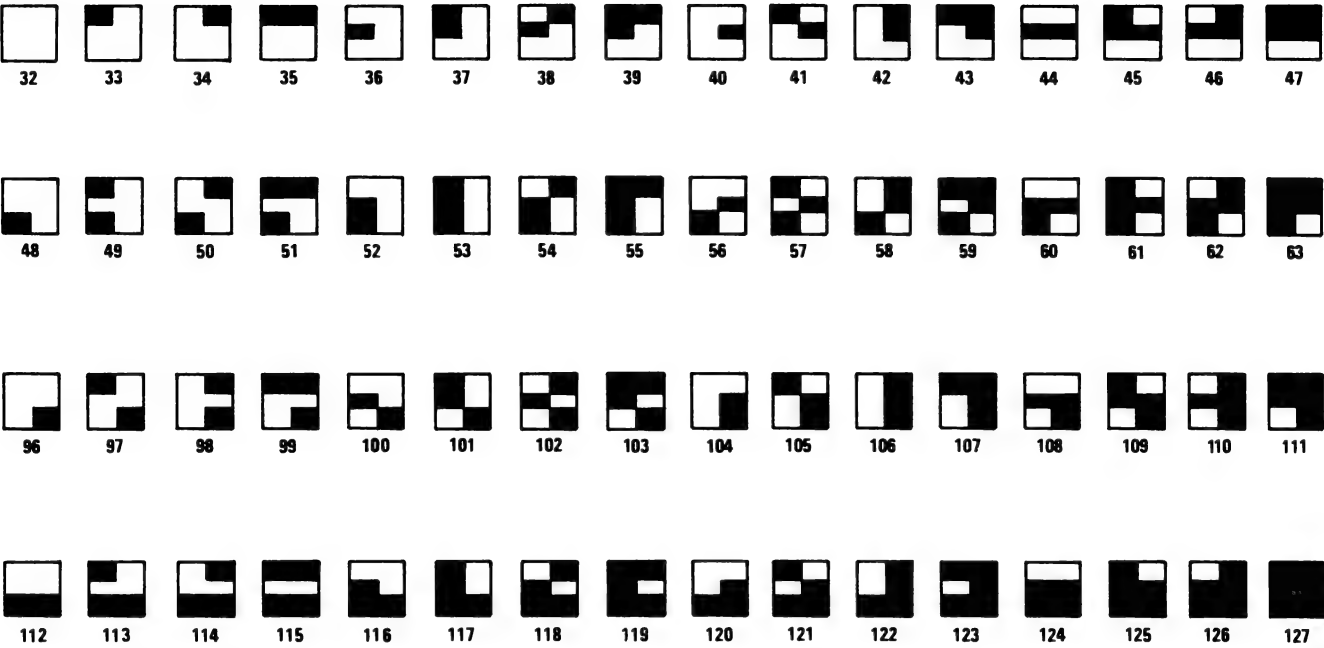
BUT : *Utiliser la police de caractères Télétel.*

Les caractères normalisés *Télétel*, tout comme les caractères ASCII disponibles sur votre ordinateur, s'inscrivent dans un carré de 8 * 8 points graphiques. Les semi-graphiques *Télétel* sont divisés en 6 zones. Chacune de ces zones est affectée à un bit d'un octet. Cet octet sera donc la représentation binaire du caractère *Télétel*.



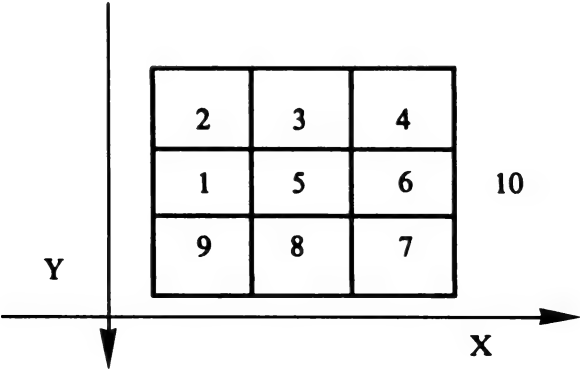
Si le bit correspondant à une zone est à 1, cette zone sera représentée, sur l'écran, en couleur d'encre. Si le bit est à \emptyset , la zone sera représentée en couleur de papier. Les bits 7 et 5 de l'octet sont invariables.

Les caractères *Télétel* : (code décimal)



Le programme que nous vous proposons pour utiliser les caractères *Télétel* consiste à lire des tables représentant des ensembles de 9 caractères choisis dans la représentation *Télétel* afin de réaliser une figure. Pour simplifier, nous avons choisi 3 lettres de l’alphabet H, A, E. Mais vous pouvez imaginer d’autres motifs au besoin en regroupant plusieurs ensembles entre eux.

Si l’on souhaite réaliser un motif à partir de caractères *Télétel*, on peut les juxtaposer sur l’axe des X mais aussi sur l’axe des Y. Comme ces caractères sont issus d’une table, il convient donc de les ordonner. Nous avons choisi l’ordre suivant :



La table DREL spécifie les déplacements relatifs que devra effectuer le curseur pour situer les caractères sur le motif.

Les valeurs de DREL sont issues du tableau ci-dessous :

Position	X	Y	Position	X	Y
1	0	0	6	1	0
2	0	-1	7	0	1
3	1	0	8	-1	0
4	1	0	9	-1	0
5	-1	1	10	3	-1

Les tables des caractères *Téléte* des trois motifs proposés (H, A, E) sont implantées à partir de l'étiquette TEXTE.

Chaque table contient 9 caractères *Téléte* plus 00 qui est le terminateur. Soit 10 octets.

Dans l'application, l'utilisateur choisit son motif à partir du clavier. Seules les touches 1, 2, 3 sont actives. La touche « espace » permet de quitter le programme.

Au code ASCII retourné par CLAV, on soustrait \$31 (SUBB #'1), afin d'avoir dans B un nombre compris entre 0 et 2, qui représentera le rang du motif dans la table. Ce rang multiplié par 10 donne la valeur qu'il faut ajouter à TEXTE pour pointer sur le début du motif.

```
SUBB  #'1
LDA   #10T
MUL
LDX   #TEXTE
LEAX  D, X
```

Quand on sauvegarde puis restitue des octets de la pile, les valeurs contenues dans la zone mémoire attribuée à la pile ne sont pas perdues. Elles sont toujours dans les mêmes octets, seul le pointeur a été modifié. On peut donc récupérer ces valeurs par un adressage indexé, par rapport au pointeur à déplacement constant. C'est cette « astuce » que nous utilisons dans TELET pour récupérer la position précédente du curseur, afin de l'ajouter aux variations relatives en X et Y.

```
SUIT  LDB      ,Y+
      ADDB     -1,U
      LDA      ,Y+
      ADDA     -2,U
      JSR      POSCUR
```

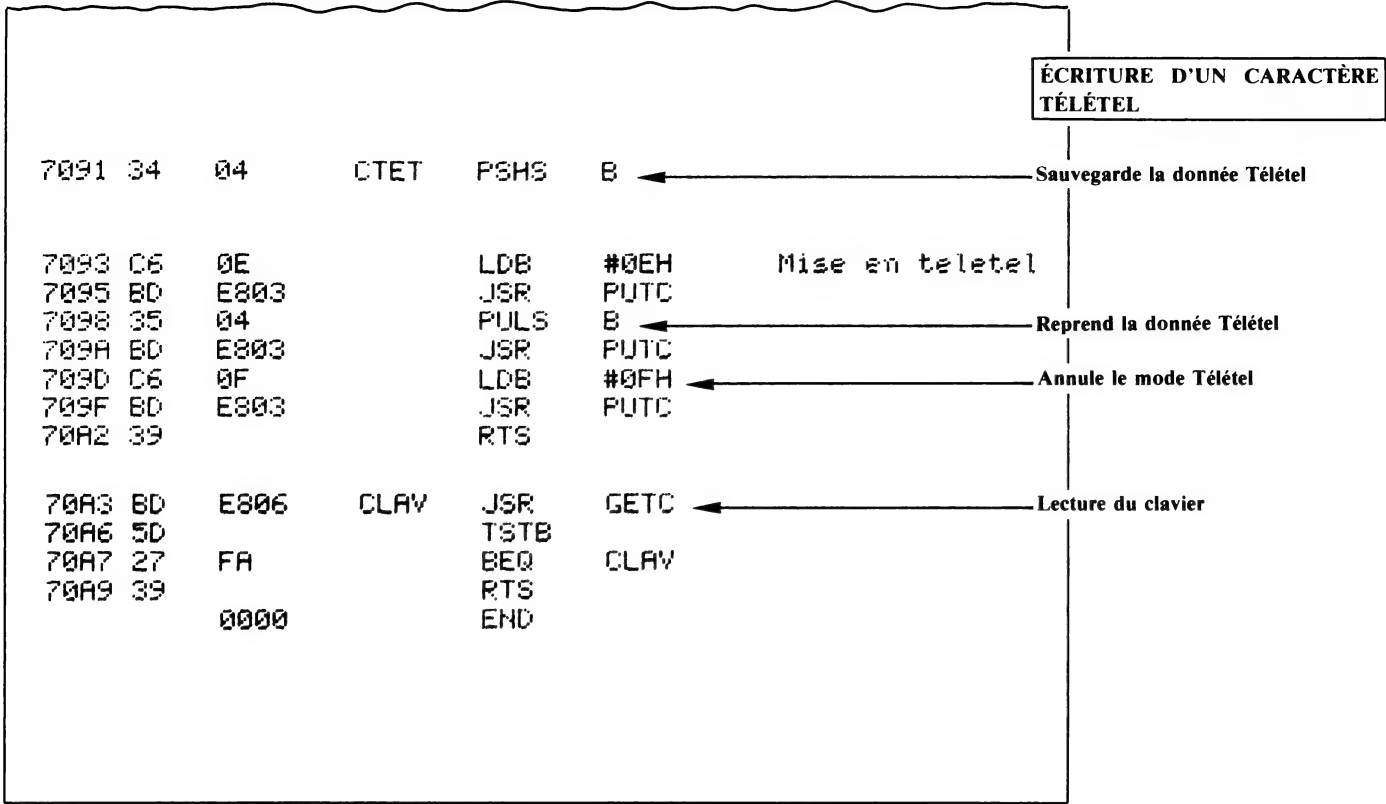
Le code \$0E permet le passage en mode *Téléte*. \$0F retourne en mode normal.

* BG ***** CARACTERES TELETEL ***** RW *

* EN ACTIVANT UNE DES TOUCHES 1-2 OU 3*
* ON GENERE UN CARACTERE DE 9 PAVES *
* CONSTRUIT AVEC DES CARACTERES TELETEL*
* LA BARRE ESPACE PERMET DE SORTIR DU *
* PROGRAMME *

	E806 E803	GETC PUTC	EQU EQU	\$E806 \$E803	Clavier Ecran	
7000 00 00 00 FF		DREL	FCB	00,00,00,-1,1,00,1,00,-1	Déplacement relatif d'un pavé à l'autre	
7004 01 00 01 00						
7008 FF						
7009 01 01 00 00			FCB	1,1,00,00,1,-1,00,-1,00		
700D 01 FF 00 FF						
7011 00						
7012 03 FF			FCB	3,-1		
7014 3D 35 6A 20		*		Premier motif	Composition des caractères	
7018 6E 20		TEXTE	FCB	61T,53T,106T,32T,110T,32T		
701A 20 6A 35 00			FCB	32T,106T,53T,00		
		*		Second motif		
701E 3D 37 23 20			FCB	61T,55T,35T,32T,32T,32T		
7022 20 20						
7024 20 70 75 00			FCB	32T,112T,117T,00		
		*		Troisième motif		
7028 35 36 69 20			FCB	53T,54T,105T,32T,106T,32T		
702C 6A 20						
702E 20 6E 3D 00			FCB	32T,110T,61T,00		
7032 CE C000		DEBUT	LDU	#ENDMEM	Pointeur Pile U	
7035 86 E3			LDA	#\$E3	Couleur courante	
7037 B7 603B			STA	\$603B	Registre COLOUR	
703A 86 0A			LDA	#10T	← Coordonnées X et Y de début de ligne	
703C C6 01			LDB	#01T		
703E BD 7079			JSR	POSCUR	← Positionne le curseur	

7041	BD	70A3	NEN	JSR	CLAV	←	Entre un caractère à partir du clavier
7044	C1	20		CMPB	#\$20		Espace?
7046	27	17		BEQ	STO		Oui
7048	C1	31		CMPB	#'1		
704A	25	F5		BLO	NEN		Plus Petit que 1
704C	C1	33		CMPB	#'3		
704E	22	F1		BHI	NEN		Plus grand que 3
7050	C0	31		SUBB	#'1		
7052	86	0A		LDA	#10T		
7054	3D			MUL	←		Multiplie par 10 pour que le registre X pointe le caractère sélectionné
7055	8E	7014		LDX	#TEXTE		
7058	30	8B		LEAX	D,X		
705A	BD	7060		JSR	TELET	←	Ecriture des caractères Télétel
705D	20	E2		BRA	NEN	←	On recommence pour écrire un autre pavé
705F	3F		STO	SWI			
							TELETEL
7060	108E	7000	TELET	LDY	#DREL	←	Pointe sur la table des déplacements relatifs
7064	E6	A0	SUIT	LDB	,Y+		
7066	EB	5F		ADDB	-1,U	←	Modification des coordonnées en ajoutant les déplacements relatifs aux valeurs précédentes
7068	A6	A0		LDA	,Y+		
706A	AB	5E		ADDA	-2,U		
706C	BD	7079		JSR	POSCUR	←	Position du curseur
706F	E6	80		LDB	,X+	←	Charge un caractère Télétel
7071	27	05		BEQ	FIN		00?
7073	BD	7091		JSR	CTET		Ecriture teletel
7076	20	EC		BRA	SUIT		
7078	39		FIN	RTS			
7079	36	06	POSCUR	PSHU	A,B	←	Positionnement du curseur
707B	C6	1F		LDB	#\$1F		
707D	BD	E803		JSR	PUTC		
7080	E6	C4		LDB	,U		
7082	C8	40		ADDB	#\$40		
7084	BD	E803		JSR	PUTC		
7087	E6	41		LDB	1,U		
7089	C8	40		ADDB	#\$40		
708B	BD	E803		JSR	PUTC		
708E	37	06		PULU	A,B		
7090	39			RTS			



41

SÉQUENCE US

BUT : *Maîtriser la séquence US.*
Utilisation de la directive d'assemblage SETDP.

Nous avons déjà utilisé ce code, notamment dans le positionnement du curseur. Mais il peut être bon de l'étudier plus à fond. Certains codes interprétables compris entre \$07 et \$1F n'ont pas une action immédiate. L'ordinateur attend d'autres données pour agir. Ce sont, en quelque sorte, les « *annonceurs* » d'autres codes.

L'ensemble de ces codes est appelé SEQUENCE.

Le code US « *annonce* » une séquence de positionnement du curseur ou de définition de la fenêtre dite « *plein écran* ».

Positionnement du curseur :

On peut mettre le curseur au début d'une ligne, ou bien à l'intersection d'une ligne et d'une colonne. Ces deux actions sont spécifiées par les quatre bits (quartet) de poids fort des données. Les poids faibles représentent la donnée elle-même. Il faut trois appels à la routine PUTC pour définir la position du curseur. Séquence que l'on peut résumer ainsi :

US/PUTC/P1/PUTC/P2/PUTC

Pour positionner le curseur en début de ligne, P1 et P2 doivent être compris entre \$30 et \$39.

P1 =

3	x
---	---

 ← dizaine de la ligne

P2 =

3	x
---	---

 ← unité de la ligne

Notez que la ligne est exprimée en décimal.

Pour placer le curseur au début de la ligne 10, il faut envoyer la séquence suivante :

US/PUTC/\$31/PUTC/\$30/PUTC.

Pour que le curseur soit au croisement d'une ligne et d'une colonne, il faut envoyer :

P1 = 40 + la ligne en code hexadécimal

P2 = 40 + la colonne en code hexadécimal

Ainsi, la séquence :

US/PUTC/\$48/PUTC/\$52/PUTC

fera apparaître le curseur sur la ligne 8 et sur la colonne 12 en hexa, soit 18 en décimal.

Définition de la fenêtre :

La fenêtre est une portion d'écran, entre une ligne dite « *ligne haute* » et une ligne dite « *ligne basse* », sur laquelle les attributs de type « *plein écran* » sont actifs. La fenêtre est définie par deux séquences US : une séquence « *ligne basse* » et une séquence « *ligne haute* ». Si une seule ligne est modifiée (haute ou basse), l'autre reste active à sa position précédente.

Trois appels à la routine PUTC sont nécessaires pour une ligne, et peuvent être matérialisés ainsi :

US/PUTC/L1/PUTC/L2/PUTC

Pour distinguer la ligne haute de la ligne basse, le quartet des poids forts de L1 et L2 devra être égal à :

1 pour la ligne basse et 2 pour la ligne haute,

le quartet des poids faibles représentant respectivement les dizaines et les unités de la ligne.

Ainsi, pour définir la ligne basse de la fenêtre sur la ligne 23, on enverra la séquence suivante :

US/PUTC/\$12/PUTC/\$13/PUTC

Pour la ligne haute en 5 :

US/PUTC/\$20/PUTC/\$25/PUTC

Le programme « SEQUENCE US » que nous vous proposons est en fait un ensemble de sous-programmes utilitaires qu'il vous suffira d'introduire dans vos applications, pour utiliser cette séquence sans problème.

Les lignes et colonnes sont passées par les accumulateurs A pour les lignes et B pour la colonne.

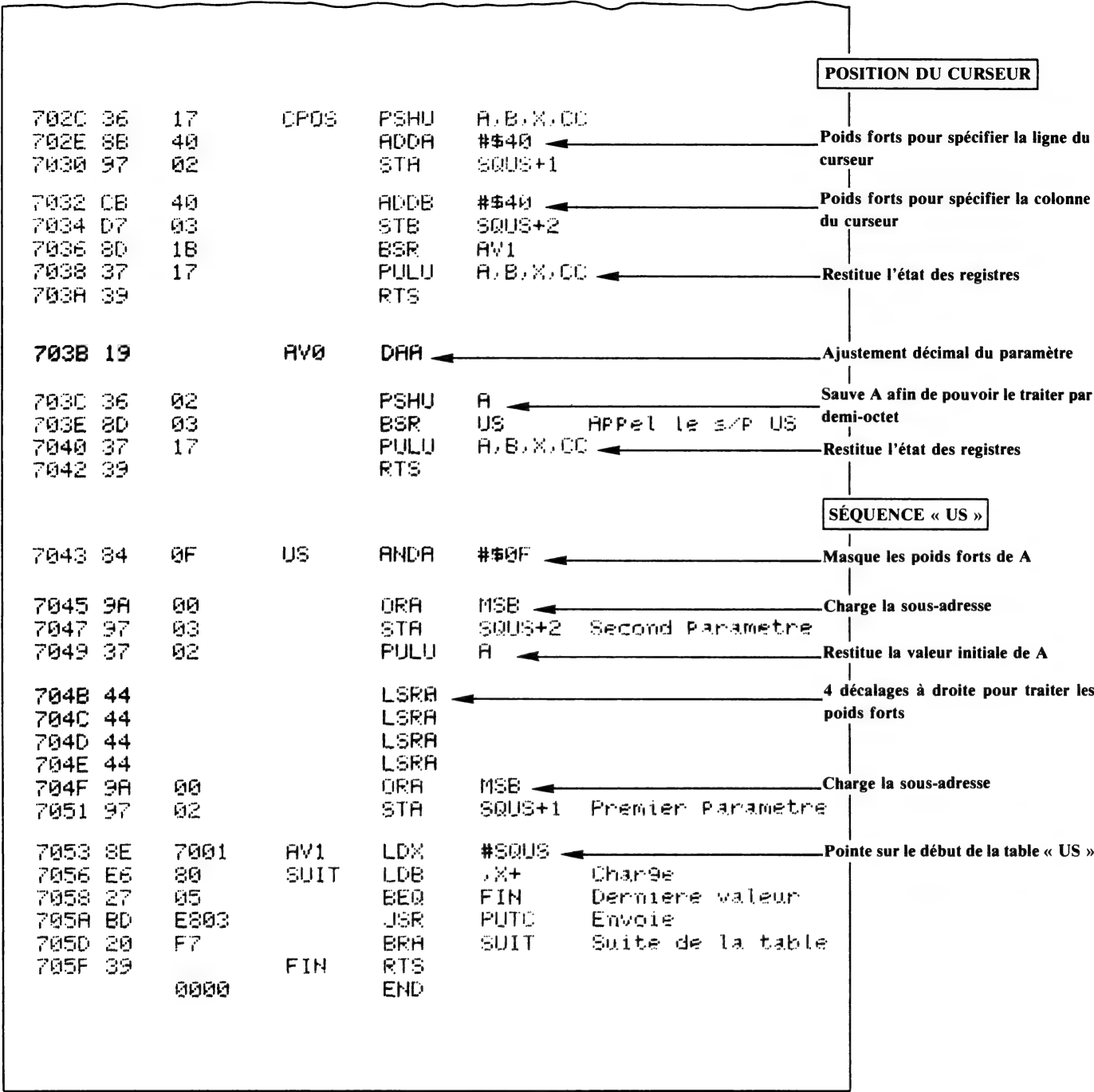
Les lignes et colonnes sont spécifiées en hexadécimal.

Les sous-programmes retournent les registres dans leur état initial, il n'est donc pas nécessaire de les sauvegarder avant d'appeler une de ces routines.

```
*****
* BG ***** SEQUENCE US ***** RW *
*****
```

```
*****
* 4 SOUS PROGRAMMES SPECIFIQUES A LA *
* SEQUENCE "US": FENB-FENH-CDLG-CPOS *
* AVEC FENB:A DOIT CONTENIR LA LIGNE *
* BASSE DE LA FENETRE *
* AVEC FENH:A DOIT CONTENIR LA LIGNE *
* HAUTE DE LA FENETRE *
* AVEC CDLG:A DOIT CONTENIR LA LIGNE *
* COURANTE SUR LAQUELLE LE CURSEUR DOIT *
* ETRE POSITIONNER *
* AVEC CPOS:A DOIT CONTENIR LA LIGNE *
* ET B LA COLONNE SUR LESQUELLES LE *
* CURSEUR DOIT ETRE POSITIONNE *
*****
```

		E803	PUTC	EGU	\$E803	Ecran	
7000				ORG	*	Origine	
	70			SETDP	*K-8		Définit la page zéro à partir des poids forts de l'adresse courante
7000			MSB	RMB	1		Pour spécifier les poids forts représentant les sous-adresses de la séquence « US »
7001	1F		SOUS	FCB	\$1F	Séquence "US"	
7002				RMB	2		
7004	00			FCB	\$00		
7005	CE	C000	DEBUT	LDU	#ENDMEM	Pointeur U	Programme pour essai
7008	86	70		LDA	*K-8		Chargement du registre de page direct « DP »
700A	1F	88		IFR	A:DP		
700C	06	0A		LDB	#\$0A	Valeurs pour essai	
700E	86	0A		LDA	#\$0A		
7010	BD	702C		JSR	CPOS		Appel de sous-programme spécifique à la séquence « US »
7013	3F			SWI			
LIGNE BASSE DE LA FENÊTRE							
7014	36	17	FENB	PSHU	A,B,X,CC		Sauvegarde les registres courants
7016	06	10		LDB	#\$10		Poids forts pour spécifier que la séquence définit la ligne basse de la fenêtre
7018	07	00		STB	MSB		
701A	20	1F		BRA	AV0		
701C	36	17	FENH	PSHU	A,B,X,CC		LIGNE HAUTE DE LA FENÊTRE
701E	06	20		LDB	#\$20		Poids forts pour spécifier que la séquence définit la ligne haute de la fenêtre
7020	07	00		STB	MSB		
7022	20	17		BRA	AV0		
7024	36	17	CDLG	PSHU	A,B,X,CC		
CURSEUR EN DÉBUT DE LIGNE							
7026	06	30		LDB	#\$30		Poids forts pour spécifier que le curseur doit être positionné en début de ligne
7028	07	00		STB	MSB		
702A	20	0F		BRA	AV0		



42

SÉQUENCE ESC

BUT : Utiliser la séquence ESC. Définir les attributs.

La séquence « ESC », que l'on appelle aussi séquence « *d'échappement* » est utilisée pour déclarer les attributs couleur de l'écran et les attributs (ATT) divers comme la mise en double taille, l'inscrustation, etc.

Ces attributs sont de deux types :

- les attributs courants
- les attributs plein écran (PE)

Qu'ils soient courants ou plein écran, les attributs ont le même code. Toutefois, certains attributs ne peuvent s'appliquer qu'au type courant, et ne sont pas reconnus en type PE. Exemples : couleur du tour, double taille, etc.

Attributs courants

L'attribut courant est l'attribut valable à un instant donné jusqu'à nouvelle définition d'un autre attribut qui annule l'attribut précédent.

C'est l'attribut qui concerne l'affichage ultérieur.

Pour définir un attribut courant, la séquence est de la forme :

ESC/PUTC/ATT/PUTC

Attribut plein écran

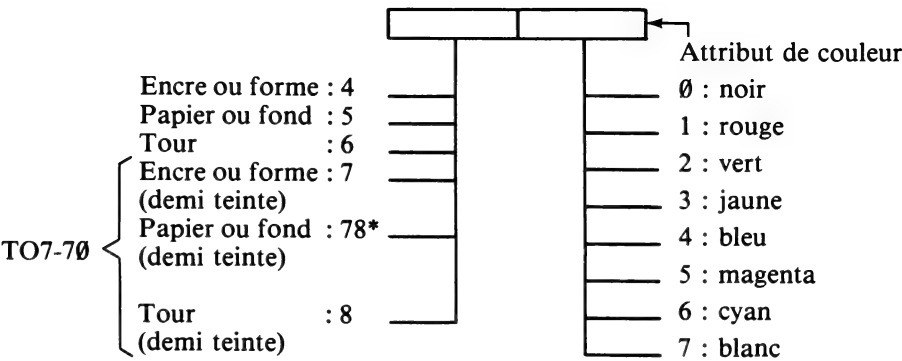
L'attribut plein écran est l'attribut qui est actif sur la zone écran définie par la fenêtre. A l'apparition de l'attribut, toute la fenêtre est modifiée.

Pour définir un attribut plein écran, la séquence est de la forme :

ESC/PUTC/\$23/PUTC/\$20/PUTC/ATT/PUTC

Attributs de couleur

Les attributs de couleur sont applicables sur la partie de l'écran sélectionnée par le quartet de poids fort de l'octet de définition de l'attribut. Le quartet de poids faible spécifie les couleurs.



* 0 1 1 1 1 X X X X

Les demi-teintes (gris, rose, vert clair, sable, bleu clair, parme, bleu ciel, orange) ne sont accessibles que sur TO7-70.

Attributs divers

Les attributs divers sont les attributs autres que les attributs de couleur.

Double hauteur : code ATT : \$4D
effet : le caractère est affiché sur deux lignes

Double largeur : code ATT : \$4E
effet : le caractère est affiché sur deux colonnes

Double taille : code ATT : \$4F
effet : le caractère est affiché sur deux lignes et deux colonnes

Taille normale : code ATT : \$4C
effet : retour aux dimensions normales d'affichage (une ligne, une colonne)

Masquage : code ATT : \$58
effet : le caractère est affiché en noir sur fond noir, c'est-à-dire invisible

Démasquage : code ATT : \$5F
effet : l'attribut permet de faire apparaître les caractères qui suivent en couleur courante d'encre et de papier s'il est passé en attribut courant. Si la déclaration de démasquage est faite en attribut plein écran, les caractères-masques de la fenêtre vont apparaître dans les couleurs précédant le masquage.

Inversion vidéo : code ATT : \$5C
effet : les couleurs d'encre et de papier sont inversées. Une nouvelle déclaration de l'attribut rétablit les couleurs d'encre (forme) et de papier (fond).

Sans couleur : code ATT : \$68
effet : le caractère est affiché à la place du caractère précédent avec les mêmes attributs. Les attributs courants sont sans effet.

Avec couleur : code ATT : \$69
 effet : retour aux règles normales d'affichages.

Incrustation : code ATT : \$6D
 effet : valide l'incrustation d'une image vidéo (TO7-70).

Non incrustation : code ATT : \$6C
 effet : suppression de l'incrustation (TO7-70).

Mode page : code ATT : \$6B
 effet : permet de gérer l'écran par page à l'intérieur de la fenêtre.

Scroll normal : code ATT : \$6A
 effet : permet un décalage, d'une ligne vers le haut, à l'intérieur de la fenêtre, lors de l'écriture d'une nouvelle ligne qui entraînerait un débordement.

Scroll lent : code ATT : \$6E
 effet : semblable au scroll, mais le décalage vers le haut a lieu ligne par ligne donc 8 fois plus lentement que le scroll normal.

Le programme utilitaire SEQUENCE ESC permet de déclarer facilement les attributs en s'affranchissant des contraintes de génération des séquences propres à chaque type d'attribut.

L'ensemble des attributs est mis en « *équate* » dans une table d'équivalence et peut donc être utilisé pour charger A, avant d'appeler la routine du type d'attribut que l'on souhaite positionner.

L'instruction :

```
LDA #TOUPT!ROUGE
```

réalise un « ou » entre l'étiquette TOUPT (\$60) et l'étiquette ROUGE (\$01) —!— afin de charger en immédiat le registre A avec la valeur \$61.

On est souvent amené à utiliser plusieurs attributs en même temps.

Une première solution consiste à charger A avec chaque attribut et appeler la routine appropriée autant de fois que nécessaire. Soit :

```
LDA #TOUDT!VERT    tour vert clair
JSR ATTCO
LDA #FONDT!MAGEN    fond parme
JSR ATTCO
LDA #MASQ           masquage
JSR ATTCO
etc.               etc.
```

Une autre solution consiste à « former » tous les attributs dans une table et les charger dans A par un adressage indexé. Soit :

```
ECRAN  FCB  TOUPT!ROUGE,FORPT!JAUNE
        FCB  FONDT!VERT,DOHAU, etc.
        FCB  $00

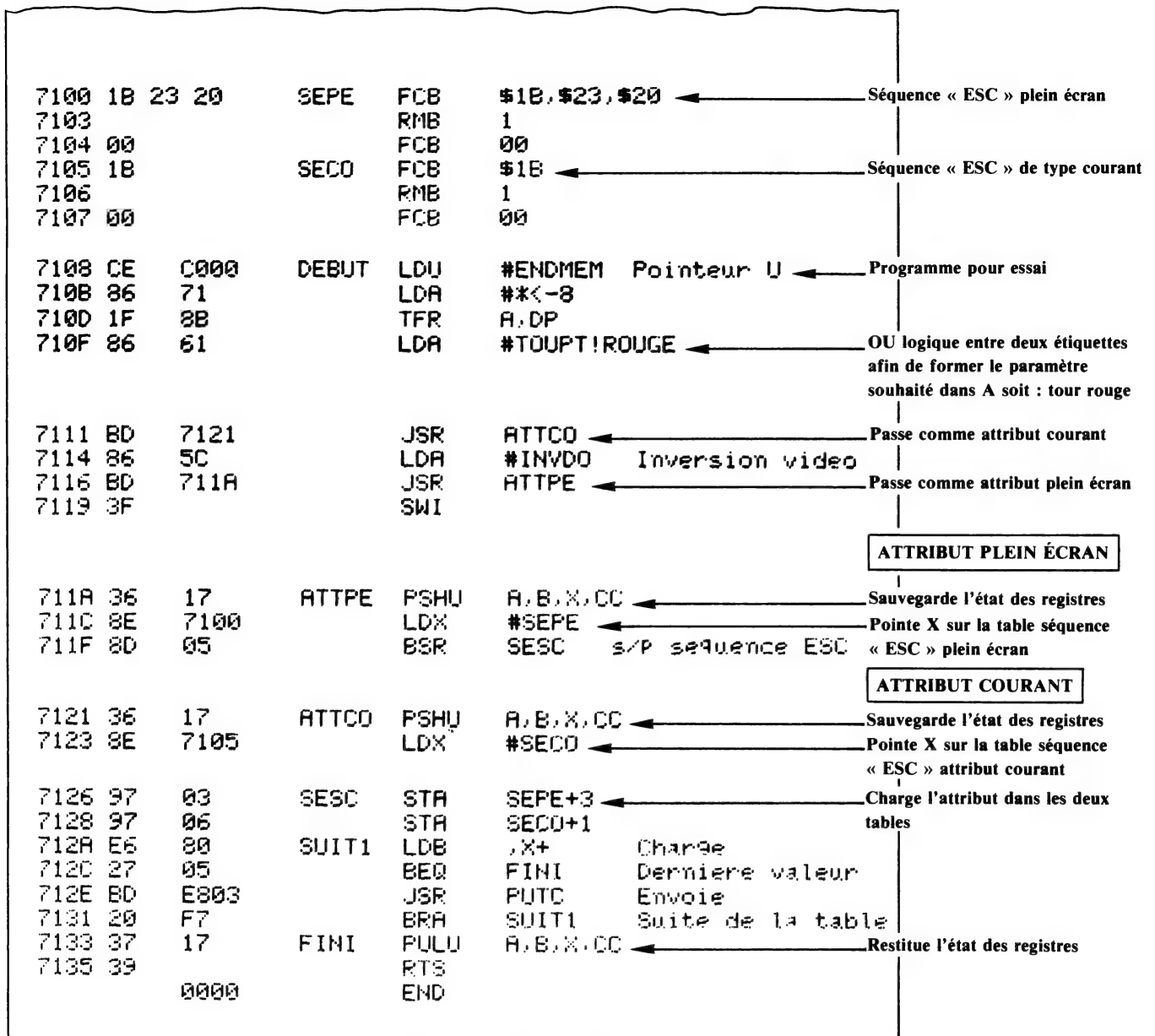
        LDX  #ECRAN
SUIT    LDA  ,X+
        BEQ  FIN
        JSR  ATTPE
        BRA  SUIT
FIN     ...
```

```
*****
* BG ***** SEQUENCE ESC ***** RW *
*****
```

```
*****
* LES ATTRIBUTS PASSES PAR LE REGISTRE *
* A SONT POSITIONNES EN ATTRIBUTS *
* COURANTS OU BIEN EN ATTRIBUTS PLEIN *
* ECRAN. *
* ATTCO PASSE LES ATTRIBUTS COURANTS *
* ATTPE PASSE LES ATTRIBUTS PLEIN ECRAN*
*****
```

7100

		ORG	*	
71		SETDP	*←-8	
E803	PUTC	EQU	\$E803	Ecran
0040	FORPT	EQU	\$40	Forme ou encre
	* Pleine teinte			
0070	FORDT	EQU	\$70	Forme ou encre
	* demie teinte			
0050	FONPT	EQU	\$50	Fond ou Papier
	* Pleine teinte			
0078	FONDT	EQU	\$78	Fond ou Papier
	* demie teinte			
0060	TOUPT	EQU	\$60	Tour Pleine teinte
0080	TOUDT	EQU	\$80	Tour demie teinte
0000	NOIR	EQU	00	Couleur: Noir
0001	ROUGE	EQU	01	" : Rouge
0002	VERT	EQU	02	" : Verte
0003	JAUNE	EQU	03	" : Jaune
0004	BLEU	EQU	04	" : Bleue
0005	MAGEN	EQU	05	" : Magenta
0006	CYAN	EQU	06	" : Cyan
0007	BLANC	EQU	07	" : Blanche
004C	TANOR	EQU	\$4C	Taille normale
004D	DOHAU	EQU	\$4D	Double hauteur
004E	DOLAR	EQU	\$4E	Double largeur
004F	DOTAI	EQU	\$4F	Double taille
0058	MASQ	EQU	\$58	Masquage
005F	DMASQ	EQU	\$5F	Demasquage
005C	INVDO	EQU	\$5C	Inversion video
0068	NMCOU	EQU	\$68	← Non modification de couleur à l'écriture d'un caractère
0069	ECOUO	EQU	\$69	← Ecriture de caractère dans la couleur courante
006C	NINCR	EQU	\$6C	← Suppression de l'incrustation « T07-70 »
006D	INCRU	EQU	\$6D	Mise en mode ← Mise en mode incrustation
006A	DEFIL	EQU	\$6A	Defilement
006B	PAGE	EQU	\$6B	mode Page
006E	DFILE	EQU	\$6E	Defilement lent



43

CRÉATION D'UN NOMBRE ALÉATOIRE

BUT : *Générer un nombre dû au hasard compris entre deux limites.*

Utilisation de l'interruption *TIMER*.

Nous vous proposons de créer un nombre aléatoire compris entre deux bornes intitulées dans le programme :

- LIHA Limite HAute
- LIBA Limite BAse

La façon la plus simple d'obtenir une valeur aléatoire consiste à exécuter un comptage rapide et de l'interrompre à un moment quelconque.

Sur votre ordinateur, le clignotement du curseur et la répétition d'une touche sont obtenus à partir de l'interruption *TIMER*, fixée à 100 ms. Cette interruption peut être aiguillée sur le programme utilisateur. Nous utiliserons cette possibilité pour interrompre le comptage, après passage dans la routine d'interruption.

Vous avez, sans aucun doute, remarqué que la fonction aléatoire était due à l'asynchronisme entre le moment du lancement de l'exécution du programme, donc du compteur, et la période des interruptions. Il est donc impossible de générer deux valeurs aléatoires à la suite l'une de l'autre, avec cette méthode, car la première interruption a pour conséquence de synchroniser le compteur et la période des interruptions. Ainsi, la seconde valeur serait une fonction de la première.

Une solution pour générer plusieurs valeurs aléatoires, avec cette méthode d'interruption, consiste à adopter un raisonnement « *parallèle* », en opposition avec la solution précédente qui pourrait être assimilée à un raisonnement « *série* ». C'est-à-dire réaliser deux compteurs qui évoluent indépendamment l'un de l'autre, entre deux limites différentes, et qui sont interrompus par la même interruption (voir le programme n° 49, BATAILLE NAVALE).

Le bit 5 du registre STATUS (\$6019) gère l'aiguillage des interruptions issues du *TIMER*.

- 0 : pas d'interruption utilisateur
- 1 : interruption utilisateur

L'adresse de la routine d'interruption utilisateur doit être mise dans le registre *TIMEPT* (\$6027, \$6028), et la routine doit être terminée par un appel au sous-programme *KBIN* qui permet de valider l'interruption pour les autres fonctions qui lui incombent normalement (curseur et clavier).

Après passage dans la routine utilisateur, il faut, bien entendu, remettre le pointeur *TIMPET* et le registre *STATUS* dans leur état initial.

 * BG *****NOMBRE ALEATOIRE***** RW *

 * GENERE DANS LE REGISTRE A UN NOMBRE *
 * ALEATOIRE COMPRIS ENTRE DEUX BORNES: *
 * LIHA ET LIBA . LE NOMBRE EST EXPRIME *
 * EN DECIMAL . *

6019 STATUS EQU \$6019 Etats des drapeaux

6027 TIMEPT EQU \$6027 ← Pointeur d'interruption utilisateur

E830 KBIN EQU \$E830 ← Sortie du programme interruption utilisateur

0019 LIHA EQU 25T ← Limite haute d'excursion du comp-
 teur

0005 LIBA EQU 5T ← Limite basse d'excursion du comp-
 teur

7300 URG \$7300 ← Origine du programme objet

7300 RES RMB 1 Etat temporaire

7301 INDIC RMB 1 ← Indicateur de passage dans la routine
 d'interruption

7302 CE 0000 DEBUT LDU #ENDMEM Pointeur U
 7305 BD 7309 JSR ALEA
 7308 3F SWI

* *****
 *****SOUS PROGRAMME*****
 *

7309 36 14 ALEA PSHU X,B Sauvegarde X et B

730B BE 6027 LDX TIMEPT ← Sauve l'état courant du pointeur uti-
 lisateur

730E 36 10 PSHU X
 7310 8E 737C LDX #INTER ← Charge l'adresse du s/p d'interrup-
 tion dans le pointeur

7313 BF 6027 STX TIMEPT
 7316 06 01 LDB #\$01 ← Positionne l'indicateur

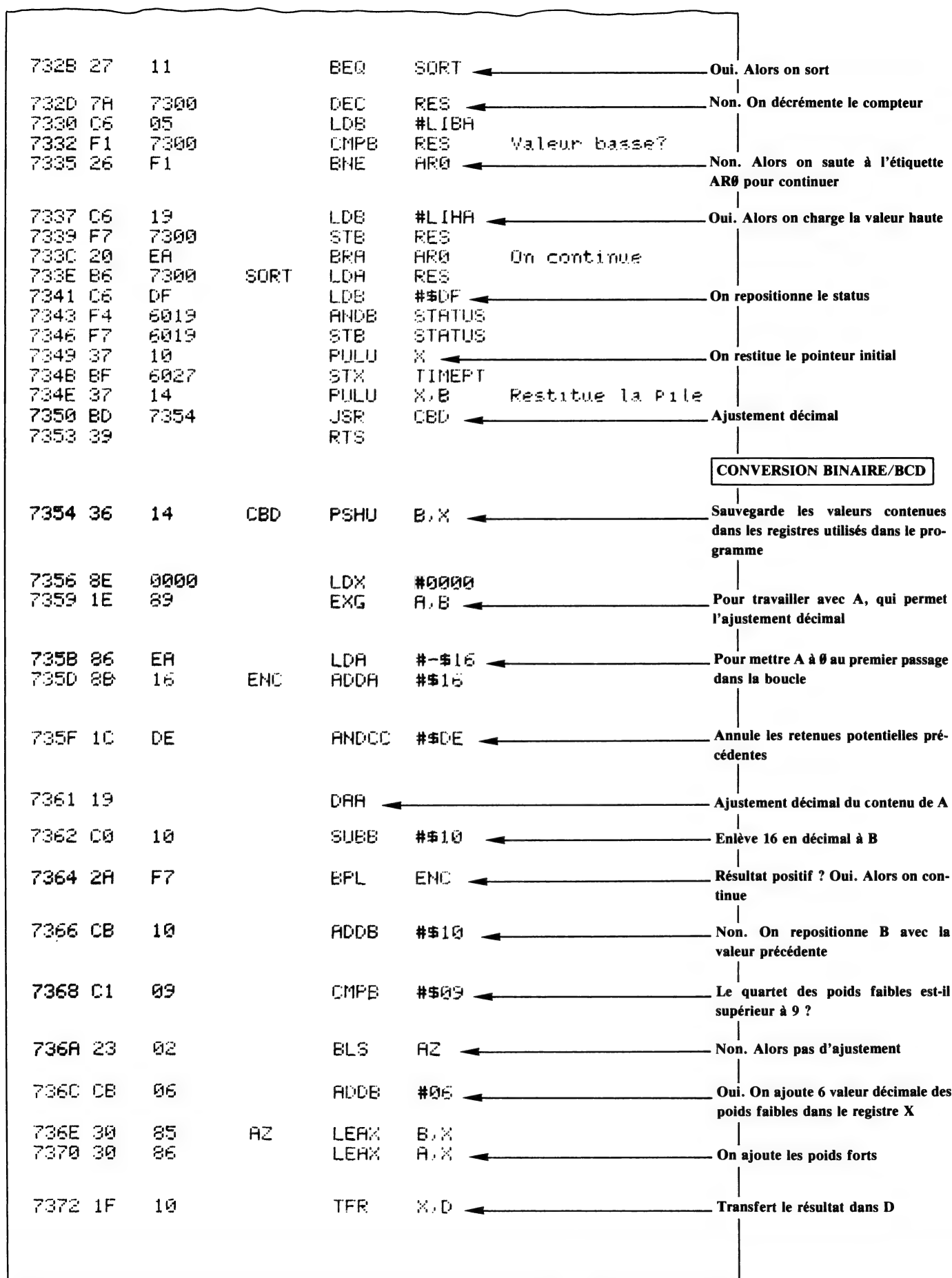
7318 F7 7301 STB INDIC

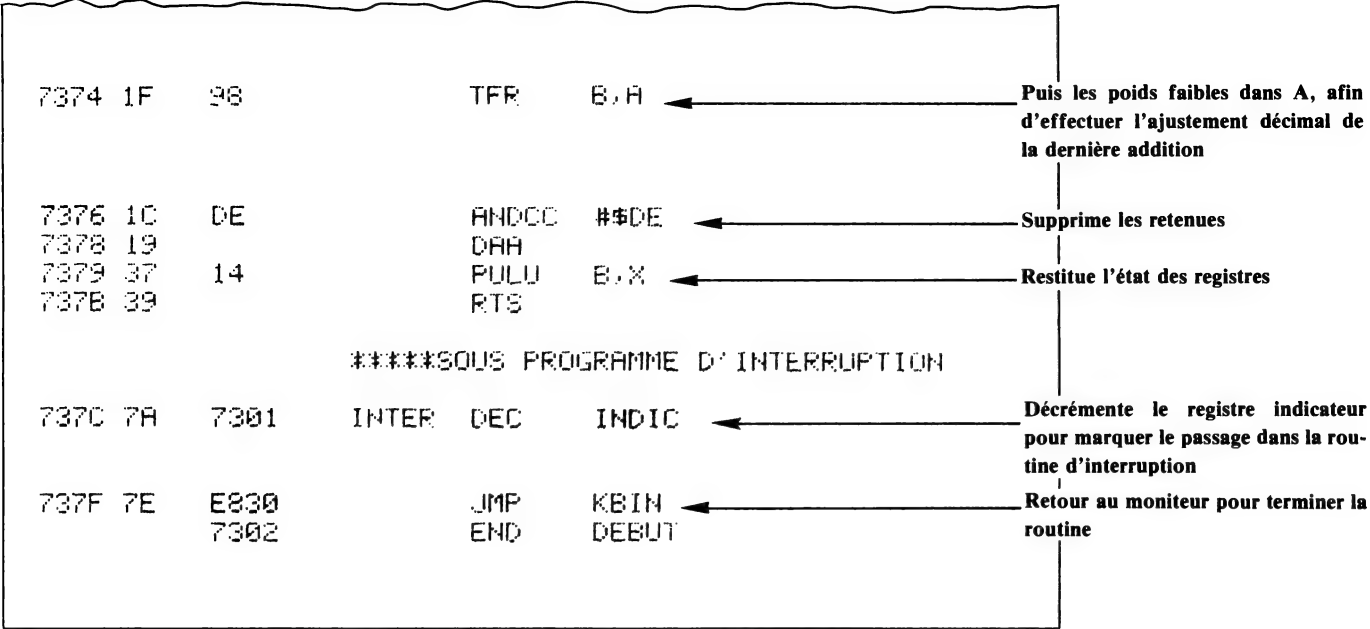
731B 06 20 LDB #\$20 ← Force le bit 5 du STATUS à 1

731D FA 6019 ORB STATUS
 7320 F7 6019 STB STATUS
 7323 06 19 LDB #LIHA ← Initialise le compteur à la valeur max

7325 F7 7300 STB RES ← Sauve le résultat partiel

7328 7D 7301 AR0 TST INDIC ← Test l'indicateur pour savoir si une
 interruption a déjà eu lieu





44

TIRAGE DU LOTO

BUT : *Utiliser le programme précédent.*

Le tirage du LOTO consiste à tirer 7 chiffres parmi 49. Nous effectuerons donc un comptage entre deux bornes initialisées à 1 et 49, que nous relancerons 7 fois après l'interruption.

Afin de conserver l'asynchronisme entre le comptage et la période des interruptions, chaque relance s'effectuera par une action sur le clavier.

Comme un chiffre ne doit sortir qu'une seule fois, le nombre contenu dans le registre au moment de l'interruption sera comparé avec les valeurs précédentes avant d'être validé.

Une temporisation, après le test du clavier, permet à l'opérateur d'avoir le temps de retirer son doigt de la touche avant qu'une boucle soit terminée.

```
*****
* BG ***** LOTO ***** RW *
*****
```

```
*****
* AFFICHE SUR L'ECRAN LE TIRAGE DU LOTO*
* UN NOUVEAU CHIFFRE APPARAÎT APRES QUE*
* L'UTILISATEUR AIT APPUYE SUR UNE *
* TOUCHE *
*****
```

6019	STATUS	EQU	\$6019	Etats des drapeaux
6027	TIMEPT	EQU	\$6027	
E830	KBIN	EQU	\$E830	
E809	KTST	EQU	\$E809	Test clavier
E803	PUTC	EQU	\$E803	Ecran
0031	LIHA	EQU	49T	Chiffres du loto
0001	LIBA	EQU	01T	
7400 1F 12 14 1F	ATTRI	FCB	\$1F,\$12,\$14,\$1F,\$20,\$20	
7404 20 20				
7406 00 1F 44 50		FCB	\$00,\$1F,\$44,\$50,\$1B,\$4F	
740A 1B 4F				

740C	1B	41		FCB	\$1B,\$41	
740E	4C	4F	54 4F	FCC	/LUTO/	
7412	1F	4B	4A 1B	FCB	\$1F,\$4B,\$4A,\$1B,\$42,\$1B	
7416	42	1B				
7418	4C	00		FCB	\$4C,\$00	
741A	1B	4C	1F 57	TANOR	FCB	\$1B,\$4C,\$1F,\$57,\$5A,\$00
741E	5A	00				
7420	1B	43	7F 1B	ENAFF	FCB	\$1B,\$43,\$7F,\$1B,\$42,\$00
7424	42	00				
7800				ORG	\$7800	
7800			RESU	RMB	8	Resultat du tirage
7808			RES	RMB	1	Etat temporaire
7809			INDIC	RMB	1	
780A			NBR	RMB	1	Nombre de chiffres à tirer
7808	0E	0000	DEBUT	LDU	#ENDMEM	Pointeur U
780E	8E	7400		LDX	#ATTRI	Attributs de départ
7811	BD	787B		JSR	MESS	
7814	8E	7800		LDX	#RESU	
7817	6F	80	AR1	CLR	/X+	Initialisation à 0 des registres RESU
7819	8C	7808		CMPL	#RESU+8	
781C	26	F9		BNE	AR1	
781E	86	07		LDA	#07	Il faut 7 chiffres
7820	B7	780A		STA	NBR	
7823	108E	7800		LDY	#RESU	Pointe sur le premier chiffre du résultat
7827	BD	7885	SUIT1	JSR	ALEA	
782A	8E	7800		LDX	#RESU	Pointe l'index de comparaison sur le premier chiffre du résultat
782D	6D	84	AR2	TST	/X	Le chiffre pointé est-il égal à zéro
782F	26	04		BNE	AV2	Non. Alors on va le comparer au dernier résultat
7831	A7	A0		STA	/Y+	Oui. Alors on le stocke
7833	20	09		BRH	AV4	On continue
7835	A1	80	AV2	CMPL	/X+	Comparaison
7837	26	F4		BNE	AR2	Si non égal. On continue
7839	7C	780A		INC	NBR	Si égal on annule ce chiffre et on continue
783C	20	1A		BRH	A02	
783E	1F	89	AV4	TFR	A:B	Pour conserver le résultat dans A
7840	54			LSRB		
CONVERSION BDC/ASCII						
7841	54			LSRB		Quartet de poids forts dans B
7842	54			LSRB		
7843	54			LSRB		
7844	0B	30		ADDB	#'0	Ajoute \$30
7846	BD	E803		JSR	PUTC	Imprime
7849	1F	89		TFR	A:B	Recharge B avec la valeur du dernier résultat
784B	04	0F		ANDB	#\$0F	Masque les poids forts
784D	0B	30		ADDB	#'0	Ajoute \$30
784F	BD	E803		JSR	PUTC	Imprime

7852 8E	7420		LDX	#ENAFF	←	Modification des attributs entre cha-
7855 8D	787B		JSR	MESS		que affichage
7858 8D	78D0	AQZ	JSR	CLAV	Test le clavier	
785B 06	07		LDB	#\$07	Bip	
785D 8D	E803		JSR	PUTC		
7860 8D	7871		JSR	TEMPO	Pour ralentir	
7863 7A	780A		DEC	NBR	←	Compte le nombre de chiffres du
						résultat
7866 27	02		BEQ	FIN	←	7 chiffres ? Oui. Alors c'est la fin
7868 20	8D		BRA	SUIT1	Non. On continue	
786A 8E	741A	FIN	LDX	#TANOR	←	Remise en taille normale
786D 8D	787B		JSR	MESS		
7870 3F			SWI			
* ***** *****SOUS PROGRAMMES***** * *****						
7871 8E	8000	TEMPO	LDX	#\$8000	←	Temporisation
7874 30	1F	CONTI	LEAX	-1,X		
7876 27	02		BEQ	FINI		
7878 20	FA		BRA	CONTI		
787A 39		FINI	RTS			
787B E6	80	MESS	LDB	,X+	←	ENVOI D'UN MESSAGE
787D 27	05		BEQ	FMES		
787F 8D	E803		JSR	PUTC		
7882 20	F7		BRA	MESS		
7884 39		FMES	RTS			
7885 36	14	ALEA	PSHU	X,B	Sauvegarde X et B	ALÉATOIRE
7887 BE	6027		LDX	TIMEPT		
788A 36	10		PSHU	X		
788C 8E	78FE		LDX	#INTER		
788F BF	6027		STX	TIMEPT		
7892 06	05		LDB	#\$05		
7894 F7	7809		STB	INDIC		
7897 06	20		LDB	#\$20		
7899 FA	6019		ORB	STATUS		
789C F7	6019		STB	STATUS		
789F 06	31		LDB	#LIHA		
78A1 F7	7808		STB	RES		
78A4 7D	7809	AR0	TST	INDIC		
78A7 27	11		BEQ	SURT		
78A9 7A	7808		DEC	RES		
78AC 06	01		LDB	#LIBA		
78AE F1	7808		CMPS	RES	Valeur basse?	
78B1 25	F1		BLO	AR0		
78B3 06	31		LDB	#LIHA		
78B5 F7	7808		STB	RES		
78B8 20	EA		BRA	AR0	On continue	
78BA B6	7808	SORT	LDA	RES		
78BD 06	DF		LDB	#\$DF		
78BF F4	6019		ANDB	STATUS		
78C2 F7	6019		STB	STATUS		

7805	37	10		PULU	X	
7807	BF	6027		STX	TIMEPT	
780A	37	14		PULU	X,B	Restitue la pile
780C	BD	7806		JSR	CBD	Ajustement decimal
780F	39			RTS		

7800	BD	E809	CLAV	JSR	KTST	
7803	24	FB		BCC	*-3	
7805	39			RTS		

CLAVIER

7806	36	14	CBD	PSHU	B,X	
7808	8E	0000		LDX	#0000	
780B	1E	89		EXG	A,B	
780D	86	EA		LDA	*-\$16	
780F	8B	16	END	ADDA	#\$16	
78E1	1C	DE		ANDCC	#\$DE	
78E3	19			DAA		
78E4	00	10		SUBB	#\$10	
78E6	2A	F7		BPL	END	
78E8	0B	10		ADDB	#\$10	
78EA	01	09		CMPE	#\$09	
78EC	23	02		BLS	AZ	
78EE	0B	06		ADDB	#06	
78F0	30	85	AZ	LEAX	B,X	
78F2	30	86		LEAX	A,X	
78F4	1F	10		TFR	X,D	
78F6	1F	98		TFR	B,A	
78F8	1C	DE		ANDCC	#\$DE	
78FA	19			DAA		
78FB	37	14		PULU	B,X	
78FD	39			RTS		

CONVERSION BINAIRE/BCD

78FE	7A	7809	INTER	DEC	INDIC	
7901	7E	E830		JMP	KBIN	
		780B		END	DEBUT	

INTERRUPTION

45

GÉNÉRATION D'UNE NOTE

BUT : Créer une note de musique avec des paramètres fixes. Apprendre à utiliser la routine NOTE.

Une note est caractérisée par sa durée et sa hauteur. La durée d'une note est spécifiée par sa forme (ronde, blanche, noire, etc.) et sa hauteur par sa position sur la portée (do, ré, mi, etc.).

La hauteur et la durée d'une note sont des valeurs relatives qui doivent être affectées d'un coefficient pour spécifier les valeurs absolues de la note à jouer.

Ces coefficients sont appelés paramètres de la note.

Votre ordinateur dispose d'une routine NOTE qui permet de jouer une note dont les paramètres sont passés par des registres. Quatre registres sont associés à cette routine.

OCTAV (16 bits \$6036 et \$6037) est le registre qui spécifie dans quelle octave la note sera jouée. Cinq octaves sont autorisées sur votre ordinateur. L'octave 1 est la plus grave et le 5 la plus aiguë. Chaque octave est associée à une valeur qu'il convient de charger dans le registre OCTAV, pour définir l'octave dans laquelle on désire jouer la note suivante.

OCTAVE	1	2	3	4	5
VALEUR	16	08	04	02	01

DUREE (16 bits \$6033 et \$6034). Le registre DUREE spécifie la durée relative de chaque note en associant une valeur à chaque type de note.

Note	Valeur	Note	Valeur
Ronde	96	Double croche	06
Blanche pointée	72	Triple Croche pointée	05
Blanche	48	Triple croche	03
Noire pointée	36	<i>Dans un triolet</i>	
Noire	24	Noire	16
Croche pointée	18	Croche	08
Croche	12	Double croche	04
Double Croche pointée	09	Triple croche	02

TEMPO (16 bits \$6031 et \$6032). Le registre TEMPO définit la vitesse à laquelle doit être jouée la partition. Le tempo et la durée relative de la note définissent la durée réelle de la note :

Durée réelle = tempo * durée relative.

Le tempo peut prendre une valeur comprise entre 1 et 255, 1 spécifiant le mouvement le plus rapide.

TIMBRE (8 bits \$6035). Le registre TIMBRE peut être chargé à une valeur comprise entre 0 et 5, et spécifie le rapport cyclique du signal représentant la note. La modification du timbre aura pour conséquence une attaque différente de la note. 0, dans le registre TIMBRE, correspond à un son continu.

Votre ordinateur peut jouer 13 notes de base (une octave) d'un do au do supérieur (baptisé ut) plus un silence. Chaque note est représentée par une valeur hexadécimale.

Note	Code	Note	Code
Silence	30	Do	31
Do dièse	32	Ré	33
Ré dièse	34	Mi	35
Fa	36	Fa dièse	37
Sol	38	Sol dièse	39
La	3A	La dièse	3B
Si	3C	Ut	3D

Pour jouer une note, il faut donc charger les registres spécifiant les paramètres, et passer par l'accumulateur B le code correspondant à la note à jouer.

Le programme suivant vous permet d'essayer la NOTE. Nous vous invitons à modifier les paramètres, ainsi que la note jouée afin de vous familiariser avec cette routine.

```
*****
* BG ***** NOTES ***** RW *
*****

*****
* JOUE UNE NOTE DONT LES PARAMETRES *
* SONT PASSES PAR LES REGISTRES: *
* OCTAV- DUREE- TEMPO- TIMBR *
*****

E81E NOTE EQU $E81E Note
6037 OCTAV EQU $6037
6034 DUREE EQU $6034
6032 TEMPO EQU $6032
6035 TIMBR EQU $6035
```

* NOTES						
	0030	SIL	EQU	\$30	Silence	
	0031	DO	EQU	\$31		
	0032	DOD	EQU	\$32	D = diese	
	0033	RE	EQU	\$33		
	0034	RED	EQU	\$34		
	0035	MI	EQU	\$35		
	0036	FA	EQU	\$36		
	0037	FAD	EQU	\$37		
	0038	SOL	EQU	\$38		
	0039	SOLD	EQU	\$39		
	003A	LA	EQU	\$3A		
	003B	LAD	EQU	\$3B		
	003C	SI	EQU	\$3C		
	003D	UT	EQU	\$3D		
6D00	C6	10	DEBUT	LDB	#16T	Octave 1
6D02	F7	6037		STB	OCTAV	
6D05	C6	60		LDB	#96T	Ronde
6D07	F7	6034		STB	DUREE	
6D0A	C6	05		LDB	#05T	Timbre 5
6D0C	F7	6035		STB	TIMBR	
6D0F	C6	0A		LDB	#10T	Tempo 10
6D11	F7	6032		STB	TEMPO	
6D14	C6	31		LDB	#00	
6D16	B0	E81E		JSR	NOTE	
6D19	3F			SWI		
		0000		END		

BOÎTE A MUSIQUE

BUT : *Utiliser la routine NOTE en nous amusant.*

Au lieu de charger un à un les registres, on peut écrire en mémoire les valeurs correspondantes, ce qui nous donnera, à la lecture, via la routine NOTE, une mélodie dont on pourra faire varier le tempo, la hauteur, ainsi que le timbre.

Dans les partitions, les informations sont codées sur 16 bits, à l'exception de la variation relative de l'octave qui est codée sur 8 bits.

Les 8 bits de poids forts représentent le registre auquel on s'adresse et les 8 bits de poids faibles la valeur du paramètre à passer.

'O (\$4F) → OCTAV
'B (\$42) → TIMBRE
'T (\$54) → TEMPO

'+ (\$2B) et '-' (\$2D) représentent les variations relatives du registre OCTAV.

Nous donnons en exemple deux chansons enfantines, qui rappelleront nos jeunes années (Oui ! pépé !), « Le roi Dagobert » et « J'ai du bon tabac ».

Pour passer de l'une à l'autre, il suffit de modifier le pointeur PARTI par PARTI2. Si vous le souhaitez, nous vous proposons d'écrire la partition ci-dessous qui est un extrait de la 9^e Symphonie de Monsieur Beethoven.

Attention au dièse à la clef !

Hymne à la joie

(Extrait 9^e Symphonie)

BEETHOVEN



* BG **** BOITE A MUSIQUE **** RW *

* JOUE "EN BOUCLE FERMEE" LE MORCEAU *
* DE MUSIQUE POINTE PAR PARTI *

E81E NOTE EQU \$E81E
6034 DUREE EQU \$6034
6037 OCTAV EQU \$6037
6032 TEMPO EQU \$6032
6035 TIMBR EQU \$6035

0030 SIL EQU \$30 Silence
0031 DO EQU \$31
0032 DOD EQU \$32 D = diese
0033 RE EQU \$33
0034 RED EQU \$34
0035 MI EQU \$35
0036 FA EQU \$36
0037 FAD EQU \$37
0038 SOL EQU \$38
0039 SOLD EQU \$39
003A LA EQU \$3A
003B LAD EQU \$3B
003C SI EQU \$3C
003D UT EQU \$3D

0060 RO EQU 96T RONDE
0048 BLP EQU 72T BLANCHE POINTEE
0030 BL EQU 48T BLANCHE
0024 NOP EQU 36T NOIRE POINTEE
0018 NO EQU 24T NOIRE
0012 CRP EQU 18T CROCHE POINTEE
000C CR EQU 12T CROCHE
0009 DCRP EQU 9T Double croche pointée
0006 DCR EQU 6T DOUBLE CROCHE
0005 TCRP EQU 5T Triple croche pointée
0003 TCR EQU 3T TRIPLE CROCHE

0059 FI EQU ^Y FIN

7400 4F 04 42 00 PARTI2 FCB ^0.4.^B.^0.^T.\$0A
7404 54 0A FCB
7406 31 0C 33 0C FCB DO,CR,RE,CR,MI,CR,DO,CR
740A 35 0C 31 0C FCB
740E 33 18 33 0C FCB RE,NO,RE,CR,MI,CR,FA,NO
7412 35 0C 36 18 FCB
7416 36 18 35 18 FCB FA,NO,MI,NO,MI,NO,DO,CR
741A 35 18 31 0C FCB
741E 33 0C 35 0C FCB RE,CR,MI,CR,DO,CR,RE,NO
7422 31 0C 33 18 FCB
7426 33 0C 35 0C FCB RE,CR,MI,CR,FA,NO,SOL,NO
742A 36 18 38 18 FCB
742E 31 30 30 18 FCB DO,BL,SIL,NO
7432 38 18 38 0C FCB SOL,NO,SOL,CR,FA,CR
7436 36 0C FCB
7438 35 18 33 0C FCB MI,NO,RE,CR,MI,CR,FA,NO

NOTES

DURÉE DES NOTES

Double croche pointée

Triple croche pointée

PARTITION — J'AI DU BON
TABAC

```

743C 35 00 36 18
7440 38 18 33 30      FCB  SOL,NO,RE,BL,SOL,NO,SOL,CR
7444 38 18 38 00
7448 36 00 35 18      FCB  FA,CR,MI,NO,RE,CR,MI,CR
744C 33 00 35 00
7450 36 18 38 18      FCB  FA,NO,SOL,NO,RE,BL,FI
7454 33 30 59

7457 4F 04 42 00  PARTI FCB  /O,4,/B,0,/T,$04 ←
7458 54 04
745D 30 18 30 18      FCB  SIL,NO,SIL,NO,SI,NO,SI,BL
7461 3C 18 3C 30
7465 3A 18 3A 30      FCB  LA,NO,LA,BL,SOL,NO,SOL
7469 38 18 38
746C 48 3A 48 3C      FCB  BLP,LA,BLP,SI,NO,UT,NO,SI
7470 18 3D 18 3C
7474 18 3A 18 38      FCB  NO,LA,NO,SOL,NO,LA,NO,LA
7478 18 3A 18 3A      FCB  BLP,LA,NO,SOL,NO,LA,NO,SI
747C 48 3A 18 38
7480 18 3A 18 3C      FCB  BL,SI,NO,SI,NO,UT,NO,/+,RE
7484 30 3C 18 3C
7488 18 3D 18 2B
748C 33
7490 18 2D 3A 30      FCB  NO,/-,LA,BL,LA,NO,LA,NO
7491 3A 18 3A 18
7495 38 18 3A 18      FCB  SOL,NO,LA,NO,SI,BL,SI,NO
7499 3C 30 3C 18
749D 3C 18 3D 18      FCB  SI,NO,UT,NO,/+,RE,NO,/-,LA
74A1 2B 33 18 2D
74A5 3A
74A6 30 3A 18 3A      FCB  BL,LA,NO,LA,BLP,LA,NO,SIL
74AA 48 3A 18 30
74AE 18 3C 18 3C      FCB  NO,SI,NO,SI,BL,LA,NO,LA,BL
74B2 30 3A 18 3A
74B6 30
74B7 38 18 38 48      FCB  SOL,NO,SOL,BLP,LA,BLP,SI
74BB 3A 48 3C
74BE 18 3D 18 3C      FCB  NO,UT,NO,SI,NO,LA,NO,SOL
74C2 18 3A 18 38
74C6 18 3A 18 38      FCB  NO,LA,NO,SOL,BLP,SOL,NO
74CA 48 38 18
74CD 30 18 30 18      FCB  SIL,NO,SIL,NO,FI
74D1 59

```

PARTITION — LE ROI DAGO-BERT

```

74D2 CE 0000  DEBUT LDU  #ENDMEM Pointeur Pile U
74D5 8E 7457  ENCOR LDY  #PARTI ← X pointe sur le début de la partition

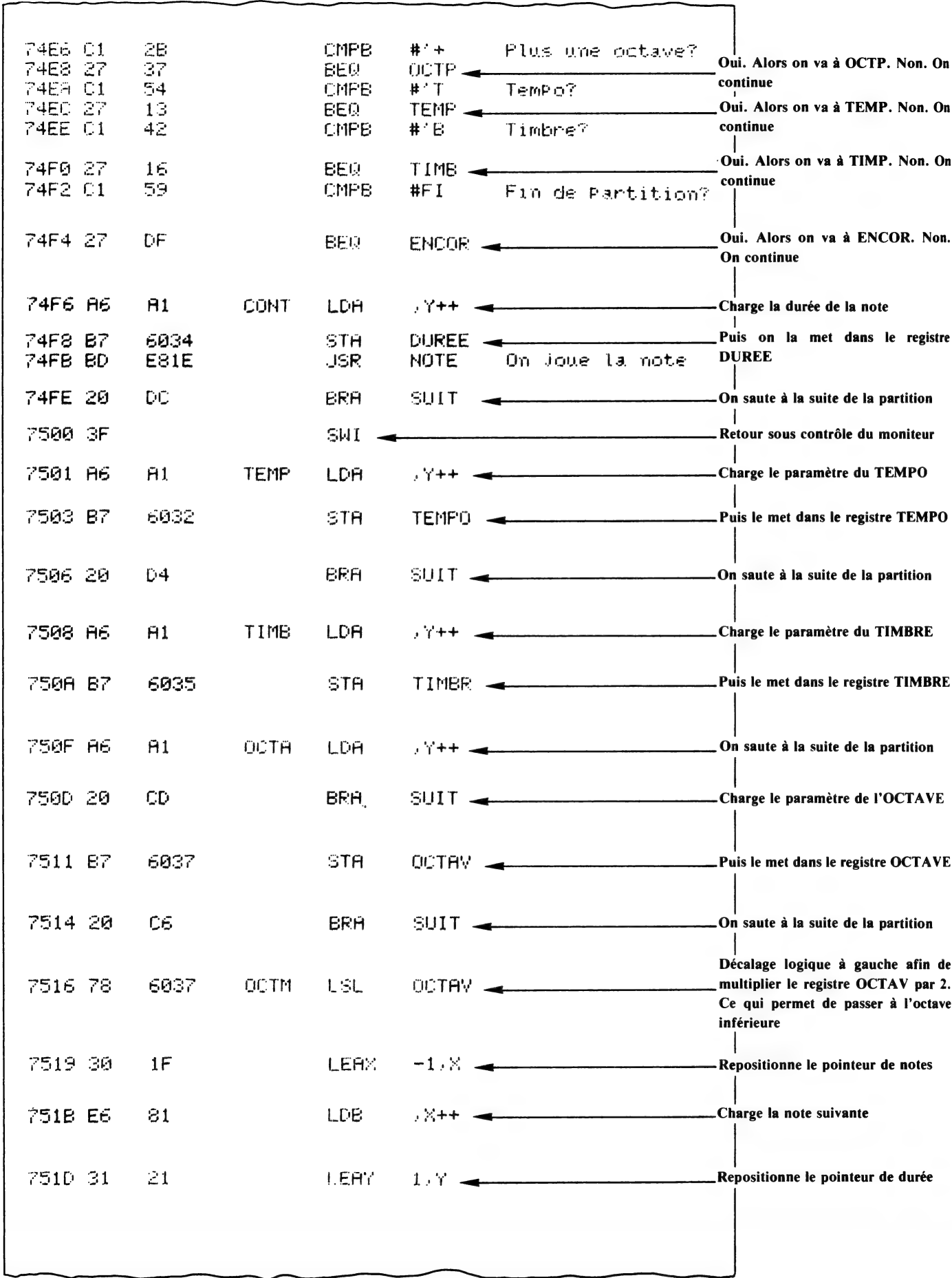
74D8 108E 7458      LDY  #PARTI+1 ← Y pointe sur le second élément de la
partition

74DC E6 81      SUIT  LDB  ,X++ ← Charge B avec la première donnée de
74DE C1 4F      CMPB  #'0      l'octave? la partition

74E0 27 2D      BEQ   OCTA ← Oui. Alors on va à OCTA. Non. On
74E2 C1 2D      CMPB  #'-      continue
Moins une octave?

74E4 27 30      BEQ   OCTM ← Oui. Alors on va à OCTM. Non. On
continue

```



751F	20	D5		BRA	CONT	
7521	74	6037	OCTP	LSR	OCTAV	
7524	30	1F		LEAX	-1,X	
7526	E6	81		LDB	,X++	
7528	31	21		LEAY	1,Y	
752A	20	CA		BRA	CONT	
		0000		END		

On continue à jouer le morceau

Décalage logique à droite afin de diviser le registre OCTAV par 2. Ce qui a pour effet de passer à l'octave supérieure

Repositionne le pointeur de notes

Charge la note suivante

Repositionne le pointeur de durée

Continue à jouer le morceau

47

CARILLON DE PORTE

BUT : *Mieux maîtriser la programmation, sous forme d'une application courante.*

A l'aide du programme précédent et d'une analyse des touches sollicitées au clavier, on peut réaliser un carillon de porte. Ce carillon jouera une mélodie qui dépendra du code demandé au clavier. On pourra ainsi savoir qui sonne à la porte en écoutant seulement la mélodie. De plus, le prénom de l'ami venant vous rendre visite sera affiché sur l'écran.

Dans votre application, nous avons retenu quatre amis qui sont : PAUL - PIERRE - JACQUES et BELLE MAMAN.

A chacun nous avons attribué un code de deux caractères :

PAUL : 13
JACQUES : Y3

PIERRE : R5
BELLE MAMAN : TE

et un air les personnalisant.

Comme Paul est toulousain, on lui jouera un « *air toulousain* ».

Pour Jacques, qui est un garçon assez sensible du porte-feuille, « *L'écossaise* » s'impose.

Pierre est un « fana » de régates en mer. Pour lui, ce sera « *Le petit navire* ».

Belle-maman est nostalgique de sa Normandie natale. Pour lui faire plaisir, nous lui jouerons « *Ma Normandie* ».

Tous les autres codes aboutiront à l'exécution « *du furet* » qui signifie que la personne n'a pas de code personnalisé.

Si le délai maximum autorisé est dépassé entre les deux caractères constituant un code permis, le programme interprètera cela comme un code non autorisé et jouera « *Le furet* ».

```
*****
* BG      ***CARILLON de PORTE***  RW *
*****
E803  PUTC  EQU    $E803
E806  GETC  EQU    $E806
E81E  NOTE  EQU    $E81E
6034  DUREE EQU    $6034
6037  OCTAVE EQU    $6037
6032  TEMPO EQU    $6032
6035  TIMBRE EQU    $6035
```

7F00	31	33	PAUL	FCC	/13/	← CODES DES AMIS
7F02				RMB	1	
7F03	52	35	PIERRE	FCC	/R5/	
7F05				RMB	1	
7F06	59	33	JACQUE	FCC	/Y3/	
7F08				RMB	1	
7F09	54	45	BELMER	FCC	/TE/	Belle Maman
7F0B				RMB	1	
7F0C			AMIS	RMB	1	Indicateur d'amis

0030	SIL	EQU	\$30	Silence	← NOTES
0031	DO	EQU	\$31		
0032	DOO	EQU	\$32	D = diese	
0033	RE	EQU	\$33		
0034	RED	EQU	\$34		
0035	MI	EQU	\$35		
0036	FA	EQU	\$36		
0037	FAD	EQU	\$37		
0038	SOL	EQU	\$38		
0039	SOLD	EQU	\$39		
003A	LA	EQU	\$3A		
003B	LAD	EQU	\$3B		
003C	SI	EQU	\$3C		
003D	UT	EQU	\$3D		

0060	RO	EQU	96T	← DURÉE DES NOTES
0048	BLP	EQU	72T	
0030	BL	EQU	48T	
0024	NOP	EQU	36T	
0018	NO	EQU	24T	
0012	CRP	EQU	18T	
000C	CR	EQU	12T	
0009	DCRP	EQU	9T	
0006	DCR	EQU	6T	
0005	TCRP	EQU	5T	
0003	TCR	EQU	3T	
0059	FI	EQU	^Y	

7F00	1F	12	24	1F	INIT	FCB	\$1F,\$12,\$24,\$1F,\$20,\$20	← Pour initialiser l'écran
7F11	20	20				FCB	\$00,\$1F,\$44,\$40,\$1B,\$40	
7F13	0C	1F	44	40		FCB	\$1B,\$53,\$1B,\$23,\$20,\$53	
7F17	1B	40				FCB	\$1B,\$63,\$1B,\$4F	
7F19	1B	53	1B	23		FCB	/CARILLON/	
7F1D	20	53				FCB	\$1F,\$50,\$50,\$1B,\$44	
7F1F	1B	63	1B	4F		FCB	\$00	
7F23	43	41	52	49		FCB	\$1B,\$62	
7F27	4C	4C	4F	4E		FCB	/PAUL/	
7F2B	1F	50	50	1B		FCB	\$00	
7F2F	44					FCB	\$1B,\$62	
7F30	00				PA	FCB	/PIERRE/	
7F31	1B	62				FCB	\$00	
7F33	50	41	55	4C		FCB	\$1B,\$62	
7F37	00				PI	FCB	/CARILLON/	
7F38	1B	62				FCB	\$00	
7F3A	50	49	45	52		FCB	\$1B,\$62	
7F3E	52	45				FCB	\$00	
7F40	00					FCB	\$1B,\$62	
7F41	1B	62			JA	FCB	\$00	

7F43	4A	41	43	51		FCC	/JACQUES/
7F47	55	45	53				
7F4A	00					FCB	\$00
7F4B	1B	62			BE	FCB	\$1B,\$62
7F4D	42	45	4C	4C		FCC	/BELLE MAMAN/
7F51	45	20	4D	41			
7F55	4D	41	4D				
7F58	00					FCB	\$00
7F59	1B	61	1B	41	IN	FCB	\$1B,\$61,\$1B,\$41
7F5D	49	4E	43	4F		FCC	/INCONNU/
7F61	4E	4E	55				
7F64	00					FCB	\$00
7F65	4F	04	50	00	TOULO	FCB	'0,4,'P,0,'T,\$08
7F69	54	00					
7F6B	30	18	30	18		FCB	SIL,NO,SIL,NO,RE,NO,SOL,NO
7F6F	33	18	38	18			
7F73	38	18	3C	0C		FCB	SOL,NO,SI,CR,LA,CR,SOL,NO
7F77	3A	0C	38	18			
7F7B	38	18	38	0C		FCB	SOL,NO,SOL,CR,LA,CR,SI,BL
7F7F	3A	0C	3C	30			
7F83	3C	18	3A	30		FCB	SI,NO,LA,BL,LA,CR,SI,CR,UT
7F87	3A	0C	3C	0C			
7F8B	3D						
7F8C	30	3D				FCB	BL,UT
7F8E	18	3C	18	3C		FCB	NO,SI,NO,SI,NO,SOL,CR,SI
7F92	18	38	0C	3C			
7F96	0C	3A	30	3A		FCB	CR,LA,BL,LA,NO,SOL,NO,SIL
7F9A	18	38	18	30			
7F9E	18	33	18	38		FCB	NO,RE,NO,SOL,NO,SOL,NO,SI
7FA2	18	38	18	3C			
7FA6	0C	3A	0C	38		FCB	CR,LA,CR,SOL,NO,SOL,NO,SOL
7FAA	18	38	18	38			
7FAE	0C	3A	0C	3C		FCB	CR,LA,CR,SI,BL,SI,NO,LA,BL
7FB2	30	3C	18	3A			
7FB6	30						
7FB7	3A	0C	3C	0C		FCB	LA,CR,SI,CR,UT,BL,'+,MI,NO
7FB8	3D	30	2B	35			
7FBF	18						
7FC0	33	18	33	18		FCB	RE,NO,RE,NO,'-,SOL,CR,SI
7FC4	2D	38	0C	3C			
7FC8	0C	3A	30	3A		FCB	CR,LA,BL,LA,NO,SOL,BLP,FI
7FCC	18	38	48	59			

PARTITION — AIR
TOULOUSAIN

7FD0	4F	04	50	00	ECOS	FCB	'0,4,'P,0,'T,\$06
7FD4	54	06					
7FD6	30	18	31	0C		FCB	SIL,NO,DO,CR,DO,CR,MI,CR
7FDA	31	0C	35	0C			
7FDE	35	0C	38	18		FCB	MI,CR,SOL,NO,SOL,CR,SOL,CR
7FE2	38	0C	38	0C			
7FE6	3D	18	3D	0C		FCB	UT,NO,UT,CR,UT,CR,SOL,BL
7FEA	3D	0C	38	30			
7FEE	30	18	35	0C		FCB	SIL,NO,MI,CR,SOL,CR,FA,CR
7FF2	38	0C	36	0C			
7FF6	35	0C	31	0C		FCB	MI,CR,DO,CR,MI,CR,RE,CR,DO
7FFA	35	0C	33	0C			
7FFE	31						
7FFF	0C	33	18	33		FCB	CR,RE,NO,RE,CR,MI,CR,RE,NO
8003	0C	35	0C	33			

PARTITION — ECOSSAISE

8007	18					
8008	30	18	31	00	FCB	SIL,NO,DO,CR,DO,CR,MI,NO
800C	31	00	35	18		
8010	35	00	35	00	FCB	MI,CR,MI,CR,SOL,NO,SOL,CR
8014	38	18	38	00		
8018	38	00	30	18	FCB	SOL,CR,UT,NO,UT,CR,UT,CR
801C	3D	00	3D	00		
8020	38	30	30	18	FCB	SOL,BL,SIL,NO,MI,CR,SOL,CR
8024	35	00	38	00		
8028	36	00	35	00	FCB	FA,CR,MI,CR,DO,CR,MI,CR,RE
802C	31	00	35	00		
8030	33					
8031	00	31	00	33	FCB	CR,DO,CR,RE,NO,RE,CR,MI,CR
8035	18	33	00	35		
8039	00					
803A	31	30	59		FCB	DO,BL,FI

803D	4F	04	50	00	NORM	FCB	'0,4,'P,0,'T,\$0A
8041	54	0A					
8043	30	18	30	18	FCB	SIL,NO,SIL,NO,SIL,CR,RE,CR	
8047	30	00	33	00			
804B	33	00	38	00	FCB	RE,CR,SOL,CR,SOL,CR,SI,CR	
804F	38	00	30	00			
8053	30	00	2B	35	FCB	SI,CR,'+',MI,CR,RE,NO,RE,NO	
8057	00	33	18	33			
805B	18						
805C	30	00	2D	33	FCB	SIL,CR,'-',RE,CR,UT,CR,SI	
8060	00	3D	00	30			
8064	00	3A	24	2B	FCB	CR,LA,NOP,'+',RE,CR,'-',SI	
8068	33	00	2D	30			
806C	00	3A	00	38	FCB	CR,LA,CR,SOL,NO,SIL,CR,RE	
8070	18	30	00	33			
8074	00	33	00	38	FCB	CR,RE,CR,SOL,CR,SOL,CR,SI	
8078	00	38	00	30			
807C	00	30	00	2B	FCB	CR,SI,CR,'+',MI,CR,RE,NO,RE	
8080	35	00	33	18			
8084	33						
8085	18	30	00	2D	FCB	NO,SIL,CR,'-',RE,CR,UT,CR	
8089	33	00	3D	00			
808D	30	00	3A	24	FCB	SI,CR,LA,NOP,'+',RE,CR,'-'	
8091	2B	33	00	2D			
8095	30	00	3A	00	FCB	SI,CR,LA,CR,SOL,NO,SIL,CR	
8099	38	18	30	00			
809D	33	00	3D	00	FCB	RE,CR,UT,CR,'+',RE,CR,'-',UT	
80A1	2B	33	00	2D			
80A5	3D						
80A6	24	33	00	30	FCB	NOP,RE,CR,SI,CR,UT,CR,SI	
80AA	00	3D	00	30			
80AE	24	33	00	3A	FCB	NOP,RE,CR,LA,CR,SI,CR	
80B2	00	30	00				
80B5	3A	24	59		FCB	LA,NOP,FI	

PARTITION — MA NORMANDIE

80B8	4F	04	50	00	NAVI	FCB	'0,4,'P,0,'T,\$0A
80BC	54	0A					
80BE	30	00	3A	00	FCB	SIL,CR,LA,CR,LA,CR,LA,CR	
80C2	3A	00	3A	00			
80C6	31	18	3A	18	FCB	DO,NO,LA,NO,LAD,CR,LA,CR	
80CA	3B	00	3A	00			
80CE	3A	18	38	00	FCB	LA,NO,SOL,CR,SOL,CR,SOL	

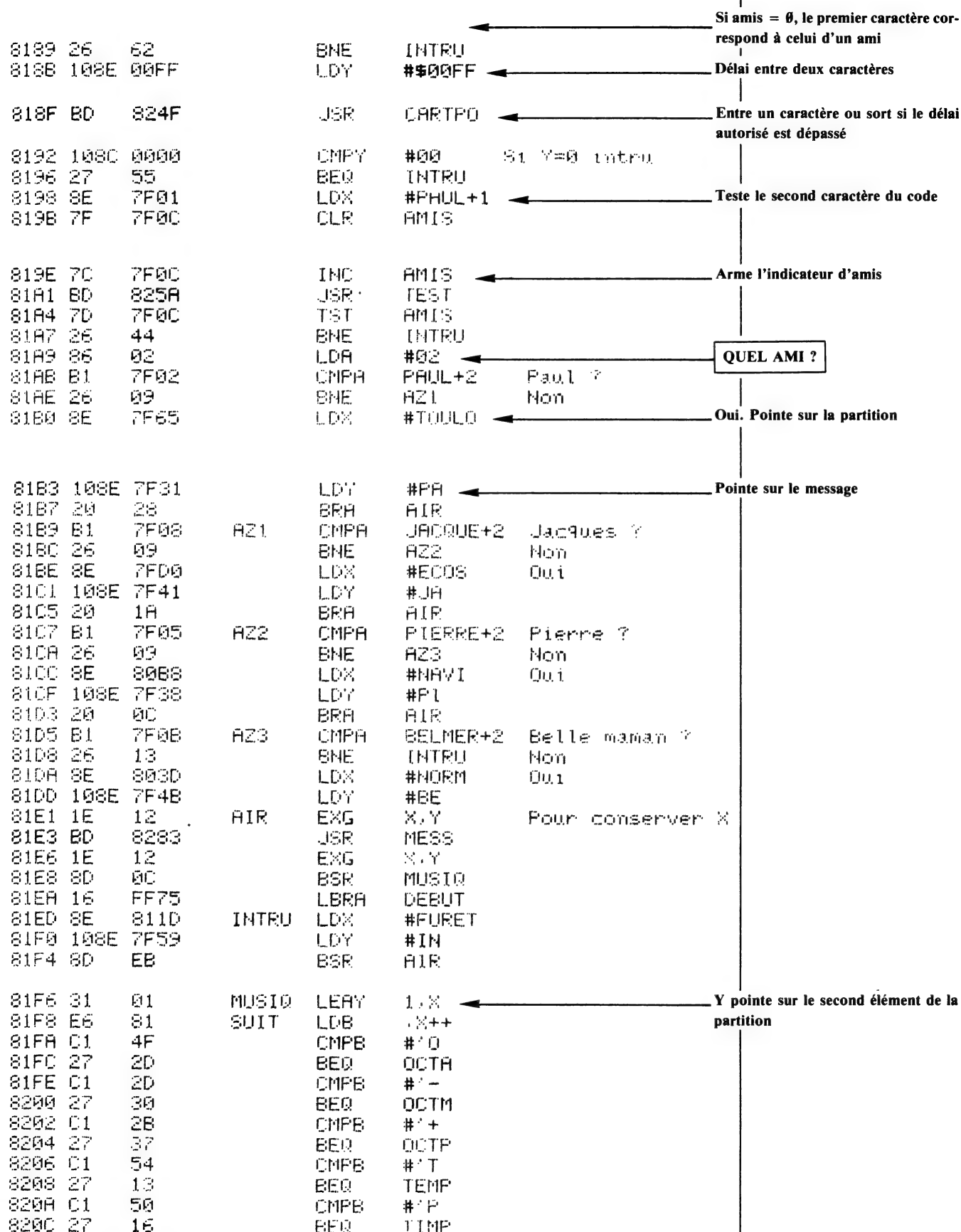
PARTITION — PETIT NAVIRE

80D2	38	0C	38		
80D5	0C	38	0C	31	FCB
80D9	18	38	18		
80DC	3A	0C	38	0C	FCB
80E0	38	18	36	0C	
80E4	3A	0C	3A	0C	FCB
80E8	3A	0C	3A	18	
80EC	3A				
80ED	18	3A	0C	3D	FCB
80F1	0C	38	0C	3A	
80F5	0C	38	0C	38	FCB
80F9	0C	38	0C		
80FC	38	0C	38	18	FCB
8100	38	18	38		
8103	0C	38	0C	3A	FCB
8107	0C	38	0C	36	
810B	0C	31	0C	36	FCB
810F	0C	3A	0C	3D	
8113	18				
8114	3D	0C	59	0C	FCB
8118	31	30	36	30	
811C	59				FCB

811D	4F	04	50	00	FURET	FCB	'0','4','P','0','T','\$0A
8121	54	0A					
8123	30	0C	38	0C		FCB	SIL,CR,SOL,CR,UT,CR,'+',RE
8127	3D	0C	2B	33		FCB	CR,MI,NO,RE,CR,RE,CR,'-'
812B	0C	35	18	33		FCB	
812F	0C	33	0C	2D		FCB	SOL,NO,UT,CR,SI,CR,LA,CR
8133	38	18	3D	0C		FCB	
8137	3C	0C	3A	0C		FCB	SOL,CR,LA,CR,SI,CR,UT,CR
813B	38	0C	3A	0C		FCB	
813F	3C	0C	3D	0C		FCB	SOL,CR,UT,CR,'+',RE,CR,MI
8143	38	0C	3D	0C		FCB	
8147	2B	33	0C	35		FCB	NO,RE,CR,RE,CR,'-',SOL,NO
814B	18	33	0C	33		FCB	
814F	0C	2D	38	18		FCB	UT,CR,SI,CR,LA,CR,SOL,CR
8153	3D	0C	3C	0C		FCB	
8157	3A	0C	38	0C		FCB	LA,CR,SI,CR,UT,NO,FI
815B	3A	0C	3C	0C		FCB	
815F	3D	18	59				

PARTITION — LE FURET

8162	0E	0000	DEBUT	LDU	#ENDMEM	Pointeur Pile II	
8165	8E	7F0D		LDX	#INIT		Efface et initialise l'écran
8168	BD	8203		JSR	MESS		
816B	7F	7F02		CLR	PAUL+2		Mise à 0 des registres
816E	7F	7F05		CLR	PIERRE+2		
8171	7F	7F08		CLR	JACQUE+2		
8174	7F	7F0B		CLR	BELMER+2		
8177	BD	8248		JSR	CAR	Entre un caractère	
817A	8E	7F00		LDX	#PAUL		Teste le premier caractère du code
817D	7F	7F0C		CLR	AMIS		
8180	7C	7F0C		INC	AMIS		Arme l'indicateur d'amis
8183	BD	825A		JSR	TEST		
8186	7D	7F0C		TST	AMIS		



820E	C1	59		CMPB	#FI
8210	27	0A		BEQ	FINI
8212	A6	A1	CONT	LDA	,Y++
8214	B7	6034		STA	DUREE
8217	B0	E81E		JSR	NOTE
821A	20	DC		BRA	SUIT
821C	39		FINI	RTS	
821D	A6	A1	TEMP	LDA	,Y++
821F	B7	6032		STA	TEMPO
8222	20	D4		BRA	SUIT
8224	A6	A1	TIMP	LDA	,Y++
8226	B7	6035		STA	TIMBRE
8229	20	CD		BRA	SUIT
822B	A6	A1	OCTA	LDA	,Y++
822D	B7	6037		STA	OCTAVE
8230	20	D6		BRA	SUIT
8232	78	6037	OCTM	LSL	OCTAVE
8235	30	1F		LEAX	-1,X
8237	E6	81		LDB	,X++
8239	31	21		LEAY	1,Y
823B	20	D5		BRA	CONT
823D	74	6037	OCTP	LSR	OCTAVE
8240	30	1F		LEAX	-1,X
8242	E6	81		LDB	,X++
8244	31	21		LEAY	1,Y
8246	20	CA		BRA	CONT
* SOUS PROGRAMMES					
8248	B0	E806	CAR	JSR	GETC
824B	5D			TSTB	
824C	27	FA		BEQ	CAR
824E	39			RTS	
824F	B0	E806	CARTPO	JSR	GETC
8252	5D			TSTB	
8253	26	04		BNE	SOR
8255	31	3F		LEAY	-1,Y
8257	26	F6		BNE	CARTPO
8259	39		SOR	RTS	
825A	E1	84	TEST	CMPB	,X
825C	26	06		BNE	AV0
825E	7C	7F02		INC	PAUL+2
8261	7F	7F0C		CLR	AMIS
8264	E1	03	AV0	CMPB	3,X
8266	26	06		BNE	AV1
8268	7C	7F05		INC	PIERRE+2
826B	7F	7F0C		CLR	AMIS
826E	E1	06	AV1	CMPB	6,X
8270	26	06		BNE	AV2
8272	7C	7F08		INC	JACQUE+2
8275	7F	7F0C		CLR	AMIS

Compte le nombre de boucles

8278	E1	09	AV2	CMPB	9,X
827H	26	06		BNE	AV3
827C	7C	7F0B		INC	BELMER+2
827F	7F	7F0C		CLP	AMIS
8282	39		AV3	RTS	
8283	E6	80	MESS	LDB	,X+
8285	27	05		BEQ	FMES
8287	8D	E803		JSR	PUTC
828A	20	F7		BRA	MESS
828C	39		FMES	RTS	
		0000		END	

48

JOURNAL LUMINEUX

BUT : *Introduction à l'animation.*

Comment animer une image vidéo issue de l'informatique ?

Voilà la question que se posent bien des informaticiens débutants.

La solution consiste, comme pour le cinéma, à envoyer successivement une suite d'images qui diffèrent légèrement les unes des autres. Grâce au phénomène de persistance rétinienne, nous aurons l'impression que les images s'animent de façon continue.

Comme application, nous vous proposons de réaliser une animation, en utilisant l'affichage de chaînes de caractères. Pour cela, nous décalerons le texte, avant chaque affichage sur l'écran, d'un caractère vers la gauche. Nous aurons, ainsi, l'impression que l'ensemble du texte se déplace de droite à gauche. Avec cette technique, nous réaliserons un journal lumineux.

Dans le programme, le texte à afficher est pointé par l'étiquette TEXTE. La chaîne sur laquelle nous travaillerons est pointée par AFF.

« AFF » est la chaîne de caractères réellement affichée et sur laquelle on effectuera le décalage. Au départ, « AFF » est constituée d'espaces.

AFF FCC / /

A chaque boucle via AR2, la chaîne AFF est translatée d'un caractère vers la gauche, par le sous-programme MOUVE.

```
AR1  LDA    1,X
      STA    ,X
      CMPX   #AFF + 18T
      BNE    AR1
```

A la sortie de MOUVE, le premier caractère de la chaîne est perdu, tandis que le dernier caractère est remplacé par le caractère pointé par le registre Y.

```
      LDA    ,Y+
      CMPA   #04
      BNE    AV1

AV1   STA    AFF + 18
```

Ce qui donne par exemple :

Boucle b → MON ORDINATEUR THOMSON ES

Boucle b + 1 → ON ORDINATEUR THOMSON EST

Si Y pointe sur la fin de TEXTE (délimiteur), les caractères sont remplacés par des espaces.

```

CMPA    #$04
BNE     AV1
LEAY    -1,Y
DEC     CESP
BNE     AV2

```

```

AV2    LDA    #$20
AV1    STA    AFF+18

```

Le registre CESP décompte les espaces entre chaque message. Le nombre d'espaces est spécifié par le registre ESP.

Entre chaque mot, la couleur d'encre est modifiée. Le registre BISTAB prend la valeur 1 ou 0. BISTAB est remis à zéro par le premier espace qui suit un caractère et mis à 1 par le second espace ou par le premier caractère du mot suivant :

texte → MON ORDINATEUR EST...



Si BISTAB est à zéro, le registre COLOR est incrémenté dans le sous-programme MCOUL, afin de modifier la couleur de forme. Si la couleur courante est blanche (\$47), on met la couleur courante en rouge (\$41).

```

*****
* BG  ****JOURNAL LUMINEUX***** RW  *
*****

```

```

*****
* AFFICHE SUR L'ECRAN LA CHAINE DE      *
* CARACTERES "TEXTE" QUI SE DEPLACE     *
* VERS LA GAUCHE. A CHAQUE MOT LA       *
* COULEUR CHANGE.                       *
* POUR SORTIR DU PROGRAMME APPUYER SUR *
* UNE TOUCHE QUELCONQUE.                *
*****

```

```

E809    KTST    EQU    $E809    Test du clavier
E803    PUTC    EQU    $E803    Affichage
E806    GETC    EQU    $E806    Clavier

```

```

7D00                                ORG    $7D00 ← Origine du programme objet

7D00 4D 4F 4E 20  TEXTE  FCC    /MON ORDINATEUR THOMSON /
7D04 4F 52 44 49
7D08 4E 41 54 45
7D0C 55 52 20 54
7D10 48 4F 4D 53
7D14 4F 4E 20
7D17 45 53 54 20          FCC    /EST FORMIDABLE      MAI/
7D1B 46 4F 52 4D
7D1F 49 44 41 42
7D23 4C 45 20 20
7D27 20 20 20 20
7D2B 4D 41 49

```

```
7D2E 53 20 53 4F          FCC      /S SON UTILISATEUR N'EST/
7D32 4E 20 55 54
7D36 49 40 49 53
7D3A 41 54 45 55
7D3E 52 20 4E 27
7D42 45 53 54
7D45 20 50 41 53          FCC      / PAS MAL NON PLUS
7D49 20 40 41 4C
7D4D 20 4E 4F 4E
7D51 20 50 40 55
7D55 53 20 20 20
7D59 20
7D5A 53 49 47 4E          FCC      /SIGNES: T05 ET M07
7D5E 45 53 3A 20
7D62 54 4F 35 20
7D66 45 54 20 4D
7D6A 4F 37 20 20
7D6E 04                    FTEXT    FCB      $04      Terminateur
7D6F 20 20 20 20          AFF      FCC      /
7D73 20 20 20 20
7D77 20 20 20 20
7D7B 20 20 20 20
7D7F 20 20 20 20
7D83 20 20 20
```

```
7D86 04                    FCB      $04      Terminateur
7D87 07                    ESP      FCB      $07      Nombre d'espaces
                        * entre chaque message
7D88                    CESP      RMB      1      Compteur d'espaces
7D89                    BISTAB    RMB      1      Bistable 1 ou 0
7D8A                    COLOR     RMB      1      Forme et rouge
```

```
7D8B 0E 0000          DEBUT    LDU      #ENDMEM Pointeur de Pile U
7D8E 86 41            LDA      #$41
7D90 97 7D8A          STA      COLOR     Forme et rouge
7D93 7F 7D89          CLR      BISTAB
7D96 7C 7D89          INC      BISTAB    Mise a 1 de BISTAB
7D99 86 7D87          LDA      ESP
7D9C 87 7D88          STA      CESP
7D9F 108E 7D00        LDY      #TEXTE    Pointe TEXTE
7DA3 BD 7E61          AR2      JSR      MOUVE    Decale la chaine
7DA6 A6 A0            LDA      ,Y+      Charge un caract.
7DAB 81 04            CMPA     #$04      Delimiteur ?
7DAA 26 13            BNE      AV1       Non
7DAC 31 3F            LEAY     -1,Y
```

```
7DAE 7A 7D88          DEC      CESP
7DB1 26 0A            BNE      AV2      Fin des espaces
7DB3 108E 7D00        LDY      #TEXTE
```

```
7DB7 B6 7D87          LDA      ESP
7DBA B7 7D88          STA      CESP
7DBD 86 20          AV2      LDA      #$20      Charge un espace
7DBF B7 7D81          AV1      STA      AFF+18
```

Charge le compteur d'espaces entre chaque message

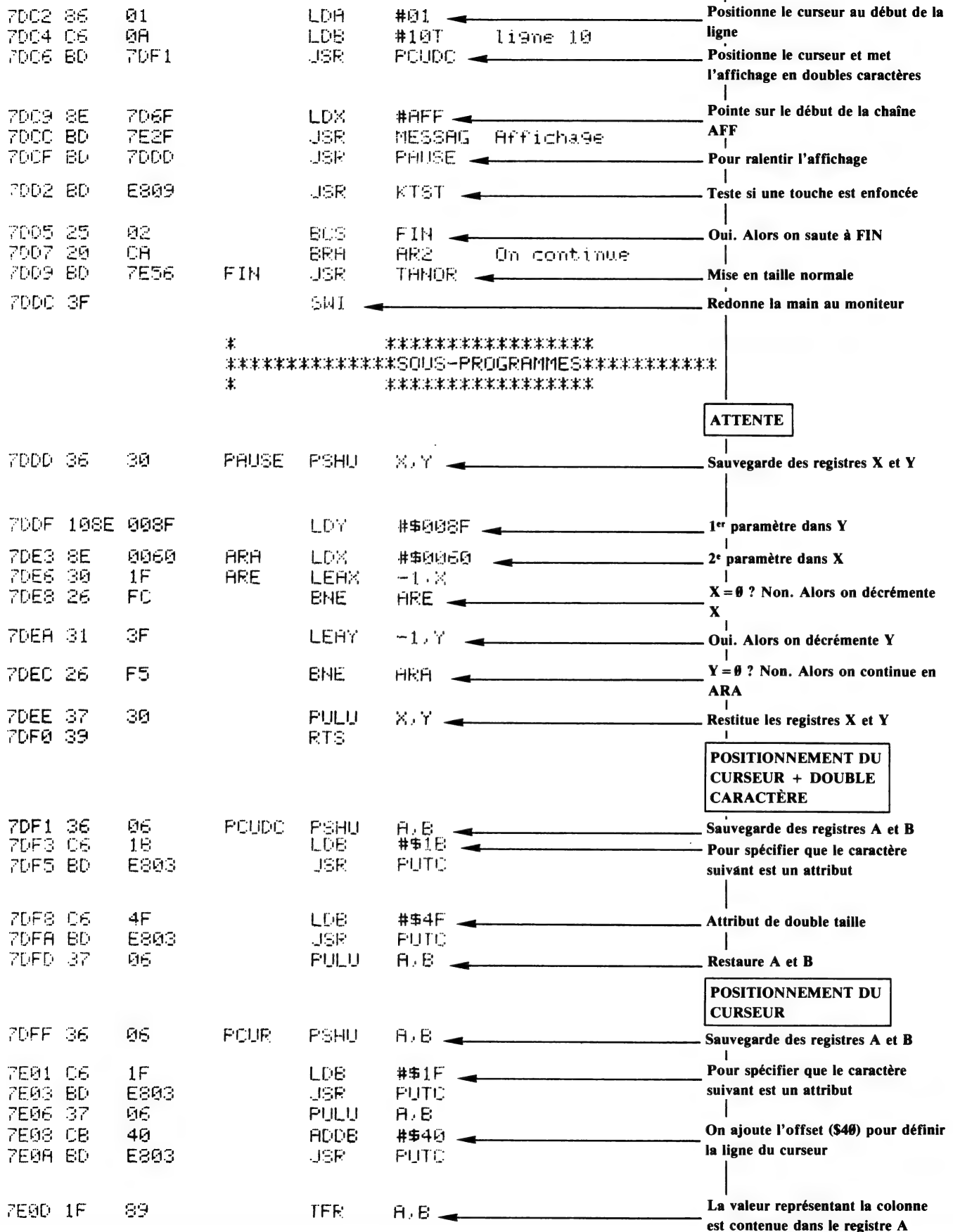
Oui. Annulation de l'incréméntation de l'index Y

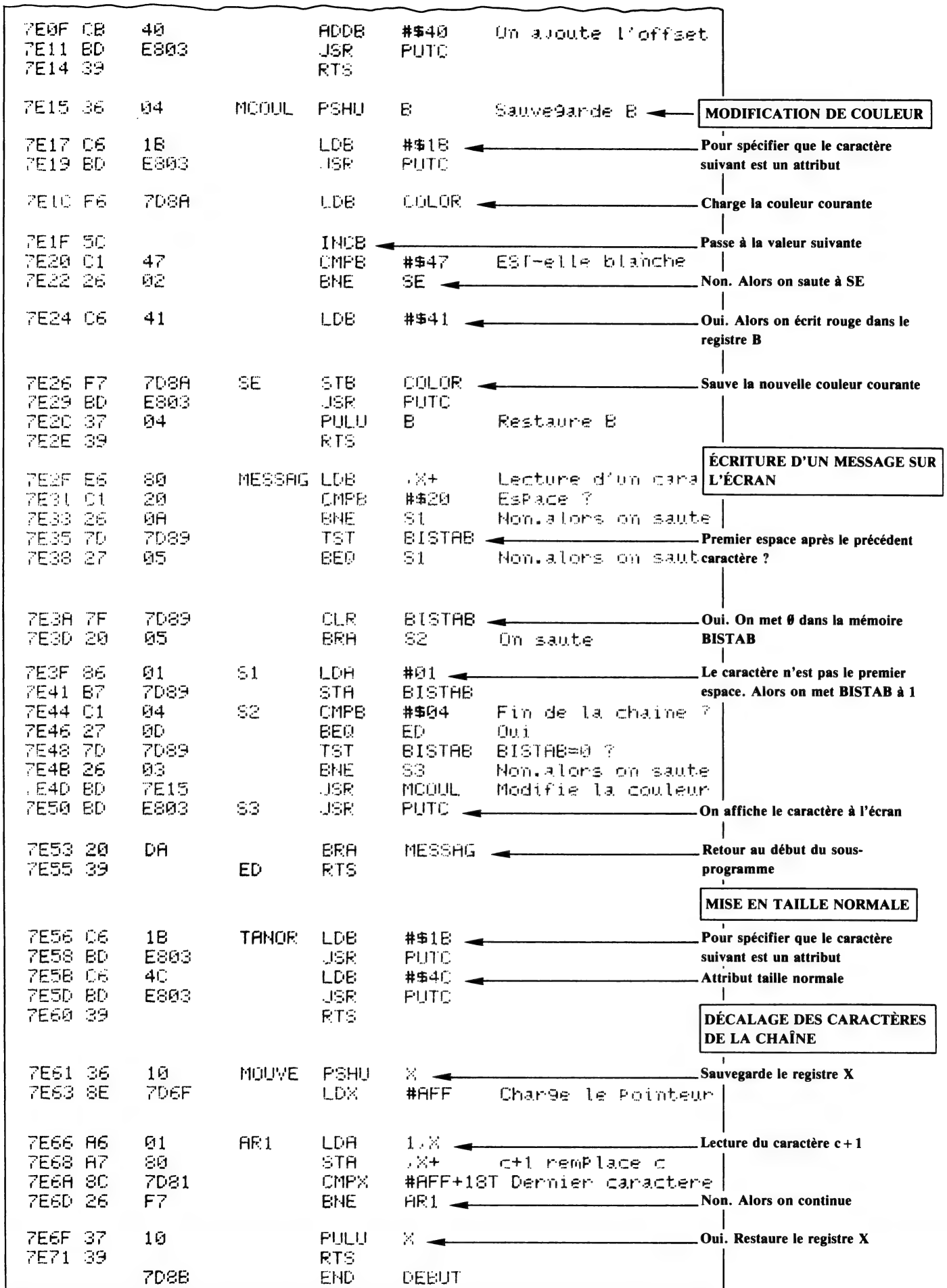
On compte les espaces entre les messages

Oui. Alors on recommence un nouveau message

Repositionne le compteur d'espaces

Stocke dans le dernier caractère de la chaîne









49

BATAILLE NAVALE

BUT : *Accroître la longueur des programmes et employer les possibilités que nous avons vues, sur une application type.*

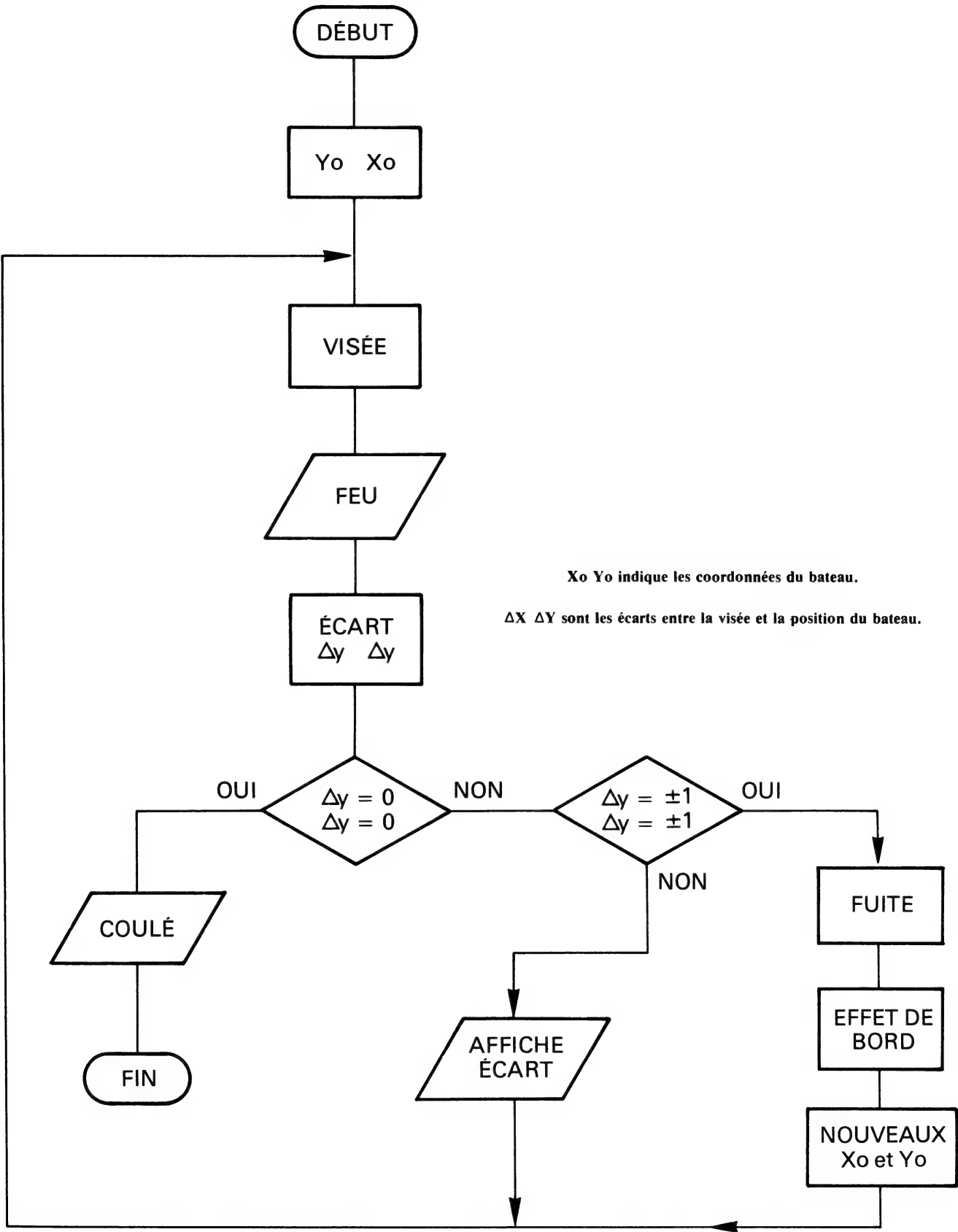
Ce programme de BATAILLE NAVALE consiste à rechercher les coordonnées d'un bateau occupant une case sur un échiquier de 29 * 16, soit 414 cases.

Les coordonnées initiales (Xo et Yo) sont établies de manière aléatoire en utilisant une routine d'interruption. Le bateau peut être visible ou invisible, il suffit pour cela de charger correctement dans le programme source le registre INVI. Pour augmenter la difficulté, le bateau « *fuit* » (s'en va) si le tir a lieu alors que la visée était sur l'une des huit cases entourant sa position. La fuite (en avant) a lieu dans la direction opposée au tir. Le bateau se déplace alors de cinq cases et a la possibilité de rebondir sur les bords (effets de bords).

Pour jouer, le programme donne la position du bateau et affiche POINTER quand cette tâche est réalisée. Vous devez alors modifier la visée, matérialisée par un carré rouge, par action sur les touches    . La touche **[ESPACE]** vous permet de déclencher le tir. Le message FEU s'affiche et un bruit de canon (ou presque) se fait entendre.

L'écart absolu minimum en X ou Y est alors affiché sur le côté droit de l'écran, puis le message POINTER est de nouveau affiché à moins que vous ne soyez tombé sur les coordonnées du bateau. Dans ce cas, le bateau est coulé et le jeu s'arrête.

Pour rejouer, taper GDEBUT.



Organigramme de la bataille navale.

```
*****
* BG *****BATAILLE NAVALE***** RW *
*****
```

```
E806 GETC EQU $E806 Clavier
E80F PLOT EQU $E80F Point
E80C DRAW EQU $E80C Trait
E803 PUTC EQU $E803 Affichage
602D USERAF EQU $602D
6019 STATUS EQU $6019 Etats des drapeaux
6027 TIMEPT EQU $6027
```

Pointeur caractère utilisateur

Pointeur d'interruption utilisateur

```
6041 CHDRAW EQU $6041
6038 FORME EQU $6038
```

Mode graphique ou caractère

Registre qui spécifie les couleurs de papier ou d'encre en mode graphique

```
6038 COLOR EQU $6038
```

Registre qui spécifie les couleurs de papier et d'encre en mode caractère

```
E830 KBIN EQU $E830
0022 LIHX EQU 34T X max
0006 LIBX EQU 6T X min
0013 LIHY EQU 19T Y max
0004 LIBY EQU 4T Y min
```

Sortie du programme interruption utilisateur

```
8900 ORG $8900
```

Origine du programme objet

```
8900 PBX RMB 1
```

Position du bateau en abscisses

```
8901 PBX RMB 1
```

Position du bateau en ordonnées

```
8902 00 PPY FCB 12T Pointeur Y
```

```
8903 12 PPX FCB 18T Pointeur X
```

```
8904 INDIC RMB 1
```

Indicateur de passage dans la routine d'interruption

```
8905 DIAGO RMB 8
```

Réserve une zone pour définir les coordonnées d'une diagonale

```
8900 DY RMB 1
```

Ecart en Y entre la visée et le bateau

```
890E DX RMB 1
```

Ecart en X entre la visée et le bateau

```
890F 20 30 31 32 TEXTE FCC 0123456789ABCDEFGHIJK A afficher sur l'écran
```

```
8913 33 34 35 36
```

```
8917 37 38 39 41
```

```
8918 42 43 44 45
```

```
891F 46 47 48 49
```

```
8923 4A 4B
```

```
8925 4C 4D 4E 4F FCC /LMNOPQRS /
```

```
8929 50 51 52 53
```

```
892D 20
```

```
892E 04 FCB $04 Terminateur
```

```
892F 20 30 31 32 TEXT2 FCC / 0123456789ABCDEF A afficher sur l'écran
```

```
8933 33 34 35 36
```

```
8937 37 38 39 41
```

```
893B 42 43 44 45
```

```
893F 46 20
```

8941	04		FCB	\$04	Terminateur	
8942	50	4F 49 4E	TEXT3	FCC	/POINTER/	
8946	54	45 52				
8949	04		FCB	\$04		
894A	20	20 46 45	TEXT4	FCC	/ FEU /	
894E	55	20 20				
8951	04		FCB	\$04		
8952	FF	81 BD A5	BATEAU	FCB	\$FF,\$81,\$BD,\$A5,\$A5,\$BD	
8956	A5	BD				
8958	81	FF	FCB	\$81,\$FF		
895A	FF	FF FF FF	POINTE	FCB	\$FF,\$FF,\$FF,\$FF,\$FF,\$FF	
895E	FF	FF				
8960	FF	FF	FCB	\$FF,\$FF		
8962	CE	C000	DEBUT	LDU	#ENDMEM	Pointeur de la pile utilisateur
8965	86	0B	LDA	#\$0B		Couleur de papier jaune, encre rouge
8967	B7	603B	STA	COLOR		
896A	86	25	LDA	#\$25	Caractere "Z"	
896C	B7	6041	STA	CHDRAW		
896F	C6	06	LDB	#LIBX		
8971	86	03	LDA	#LIBY-1		
8973	BD	8B0A	JSR	POSCUR		
8976	8E	890F	LDX	#TEXTE	Pointe sur le début du message	
8979	BD	8AFA	JSR	MESSAH	Ecrit les coordonnées horizontales	
897C	C6	06	LDB	#LIBX		
897E	86	14	LDA	#LIHY+1		
8980	BD	8B0A	JSR	POSCUR		
8983	8E	890F	LDX	#TEXTE	Pointe sur le début du message	
8986	BD	8AFA	JSR	MESSAH	Ecrit les coordonnées horizontales	
8989	C6	06	LDB	#LIBX		
898B	86	03	LDA	#LIBY-1		
898D	BD	8B0A	JSR	POSCUR		
8990	8E	892F	LDX	#TEXT2	Pointe sur le début du message	
8993	BD	8AE4	JSR	MESSAV	Ecrit les coordonnées verticales	
8996	C6	24	LDB	#LIHX+2		
8998	86	03	LDA	#LIBY-1		
899A	BD	8B0A	JSR	POSCUR		
899D	8E	892F	LDX	#TEXT2	Pointe sur le début du message	
89A0	BD	8AE4	JSR	MESSAV	Ecrit les coordonnées verticales	
89A3	BD	8B51	JSR	ALEA		
89A6	C6	02	LDB	#02		
89A8	86	01	LDA	#01		
89AA	BD	8B0A	JSR	POSCUR		
89AD	8E	8942	LDX	#TEXT3		
89B0	BD	8AFA	JSR	MESSAH		
89B3	BD	8A29	JSR	WISEE		
89B6	B6	8902	LDA	PPY	Calcul les ecarts	
89B9	B0	8900	SUBA	PBY		
89BC	B7	8900	STA	DY		
89BF	F6	8903	LDB	PPX		
89C2	F0	8901	SUBB	PBX		
89C5	F7	890E	STB	DX		
89C8	C1	01	CMFB	#01		
89CA	26	04	BNE	X1		

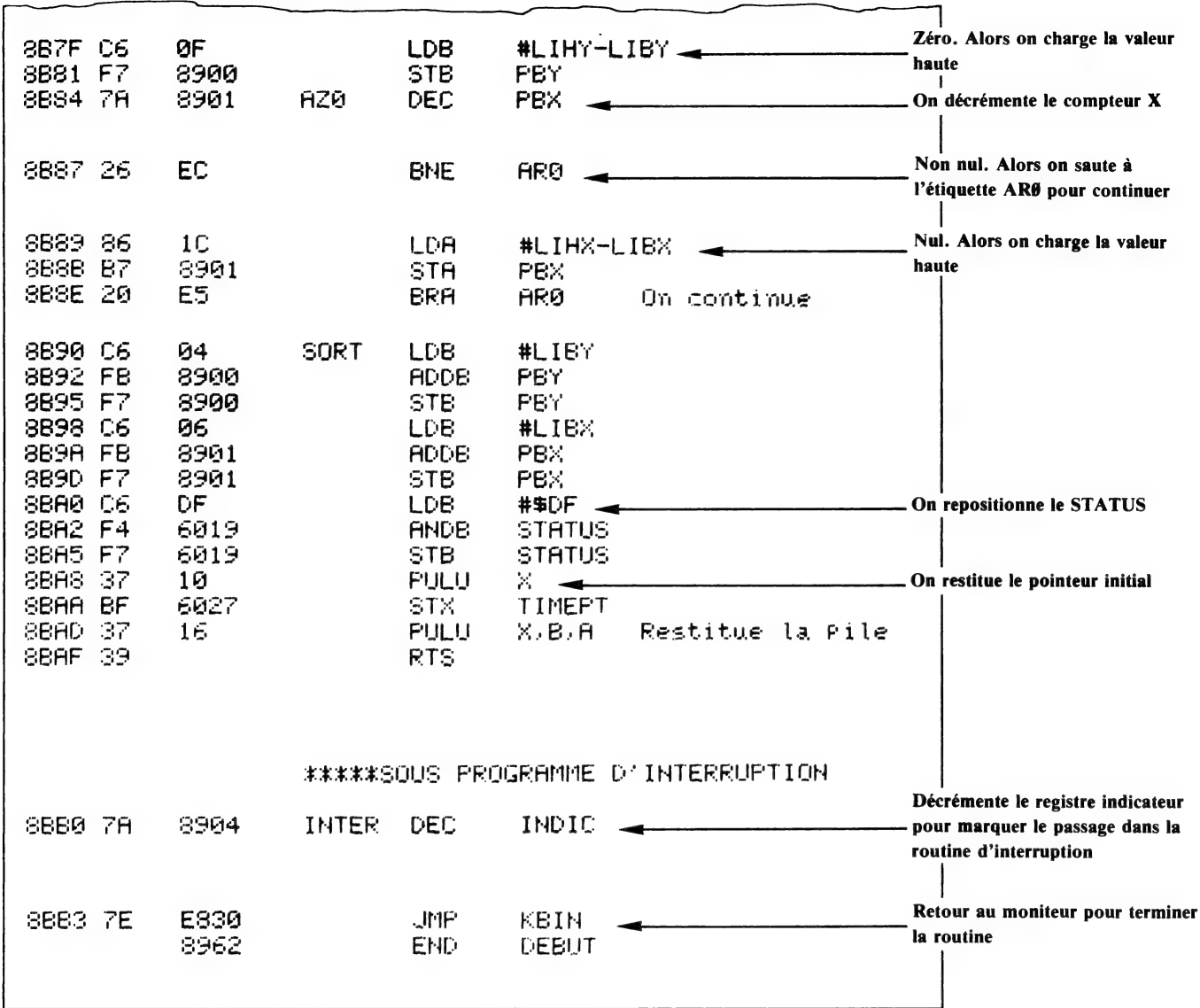
8900	06	FB		LDB	#-5
890E	20	09		BRA	X2
8900	01	FF	X1	CMPE	#-1
8902	26	04		BNE	X3
8904	06	05		LDB	#5
8906	20	01		BRA	X2
8908	5F		X3	CLRB	
8909	81	01	X2	CMPE	#01
890B	26	04		BNE	Y1
890D	86	FB		LDA	#-5
890F	20	09		BRA	Y2
89E1	81	FF	Y1	CMPE	#-1
89E3	26	04		BNE	Y3
89E5	86	05		LDA	#5
89E7	20	01		BRA	Y2
89E9	4F		Y3	CLRA	
89EA	36	06	Y2	PSHU	A,B
89EC	F6	8900		LDB	PBY
89EF	40		TR1	TSTA	
89F0	27	13		BEQ	SUIT
89F2	2A	08		BPL	POS
89F4	4C			INCA	
89F5	5A			DECB	
89F6	01	05		CMPE	#LIBY+1
89F8	26	F5		BNE	TR1
89FA	20	06		BRA	TV2
89FC	4A		POS	DECA	
89FD	5C			INCB	
89FE	01	12		CMPE	#LIHY-1
8A00	26	ED		BNE	TR1
8A02	40		TV2	NEGA	
8A03	20	EA		BRA	TR1
8A05	F7	8900	SUIT	STB	PBY
8A08	37	06		PULU	A,B
8A0A	B6	8901		LDA	PBX
8A0D	5D		RT1	TSTB	
8A0E	27	13		BEQ	SUIT1
8A10	2A	08		BPL	POS1
8A12	5C			INCB	
8A13	4A			DECA	
8A14	81	07		CMPE	#LIBX+1
8A16	26	F5		BNE	RT1
8A18	20	06		BRA	VT2
8A1A	5A		POS1	DECB	
8A1B	4C			INCA	
8A1C	81	21		CMPE	#LIHX-1
8A1E	26	ED		BNE	RT1
8A20	50		VT2	NEGB	
8A21	20	EA		BRA	RT1
8A23	B7	8901	SUIT1	STA	PBX
8A26	16	FF8A		LBRA	ENCOR

EFFETS DE BORDS

* *****					
*****SOUS PROGRAMME*****					
* *****					
8A29 BD	8A8C	VISEE	JSR	SURFA	
8A2C BD	8AA0		JSR	AFBA	
8A2F BD	8A8B		JSR	AFVI	
8A32 BD	8AB5		JSR	CLAV	
8A35 C1	09		CMPB	#\$09	← Flèche droite
8A37 26	0D		BNE	S1	
8A39 7C	8903		INC	PPX	
8A3C B6	8903		LDA	PPX	
8A3F 81	22		CMPA	#LIHX	
8A41 23	E6		BLS	VISEE	
8A43 7A	8903		DEC	PPX	
8A46 C1	08	S1	CMPB	#\$08	← Flèche gauche
8A48 26	0D		BNE	S3	
8A4A 7A	8903		DEC	PPX	
8A4D B6	8903		LDA	PPX	
8A50 81	07		CMPA	#LIBX+1	
8A52 24	D5		BHS	VISEE	
8A54 7C	8903		INC	PPX	
8A57 C1	0A	S3	CMPB	#\$0A	
8A59 26	0D		BNE	S5	← Flèche haute
8A5B 7C	8902		INC	PPY	
8A5E B6	8902		LDA	PPY	
8A61 81	13		CMPA	#LIHY	
8A63 23	C4		BLS	VISEE	
8A65 7A	8902		DEC	PPY	
8A68 C1	0B	S5	CMPB	#\$0B	← Flèche base
8A6A 26	0D		BNE	S7	
8A6C 7A	8902		DEC	PPY	
8A6F B6	8902		LDA	PPY	
8A72 81	04		CMPA	#LIBY	
8A74 24	B3		BHS	VISEE	
8A76 7C	8902		INC	PPY	
8A79 C1	20	S7	CMPB	#\$20	← Barre espace
8A7B 26	AC		BNE	VISEE	
8A7D C6	02		LDB	#02	
8A7F 86	01		LDA	#01	
8A81 BD	8B0A		JSR	POSCUR	
8A84 8E	894A		LDX	#TEXT4	
8A87 BD	8AFA		JSR	MESSAH	
8A8A 39			RTS		
8A8B F6	8903	AFVI	LDB	PPX	
8A8E B6	8902		LDA	PPY	
8A91 BD	8B0A		JSR	POSCUR	
8A94 8E	8952		LDX	#BATEAU	
8A97 BF	602D		STX	USERAF	
8A9A C6	81		LDB	##81	
8A9C BD	E803		JSR	PUTC	
8A9F 39			RTS		
8AA0 F6	8901	AFBA	LDB	PBX	
8AA3 B6	8900		LDA	PBY	
8AA6 BD	8B0A		JSR	POSCUR	
8AA9 8E	8952		LDX	#BATEAU	
8AAC BF	602D		STX	USERAF	
8AAF C6	80		LDB	#\$80	
8AB1 BD	E803		JSR	PUTC	

8AB4	39			RTS		
8AB5	BD	E806	CLAV	JSR	GETC	
8AB8	5D			TSTB		
8AB9	27	FA		BEQ	CLAV	
8ABB	39			RTS		
8ABC	8E	0030	SURFA	LDX	#LIBX*8	Chargement des coordonnées graphiques X0, Y0, X1, Y1
8ABF	BF	8905		STX	DIAG0	
8AC2	8E	0020		LDX	#LIBY*8	
8AC5	BF	8907		STX	DIAG0+2	
8AC8	8E	0110		LDX	#LIHX*8	
8ACB	BF	8909		STX	DIAG0+4	
8ACE	8E	0098		LDX	#LIHY*8	
8AD1	30	08		LEAX	8,X	
8AD3	BF	890B		STX	DIAG0+6	
8AD6	86	F9		LDA	#-\$7	Couleur de papier : cyan
8AD8	B7	6038		STA	FORME	
8ADB	86	00		LDA	#\$00	Mise en mode graphique
8ADD	B7	6041		STA	CHDRAW	
8AE0	BD	8B22		JSR	RPLGR Trace la surface	Trace la surface
8AE3	39			RTS		
8AE4	36	06	MESSAV	PSHU	B,A I Sauvegarde de B	
8AE6	E6	80		LDB	,X+	Charge un caractère ASCII dans B et incrémente le pointeur
8AE8	C1	04		CMPE	#\$04	Le dernier caractère entré est-il le terminateur ?
8AEA	27	0B		BEQ	TR	Oui. Alors on saute à la fin du programme
8AEC	BD	E803		JSR	PUTC	Non. Alors on l'imprime sur l'écran
8AEF	37	06		PULU	A,B	
8AF1	4C			INCA		
8AF2	BD	8B0A		JSR	POSDUR	
8AF5	20	ED		BRA	MESSAV	Puis on retourne au début pour lire un nouveau caractère
8AF7	37	06	TR	PULU	B,A	Restitue l'état du registre B
8AF9	39			RTS		Retour au programme principal
8AFA	36	04	MESSAH	PSHU	B Sauvegarde de B	
8AFC	E6	80		LDB	,X+	Charge un caractère ASCII dans B et incrémente le pointeur
8AFE	C1	04		CMPE	#\$04	Le dernier caractère entré est-il le terminateur ?
8B00	27	05		BEQ	*+7	Oui. Alors on saute à la fin du programme
8B02	BD	E803		JSR	PUTC	Non. Alors on l'imprime sur l'écran
8B05	20	F5		BRA	*-9	Puis on retourne au début pour lire un nouveau caractère
8B07	37	04		PULU	B	Restitue l'état du registre B
8B09	39			RTS		Retour au programme principal

8B0A	36	06	POSCUR	PSHU	A,B	
8B0C	C6	1F		LDB	#\$1F	
8B0E	BD	E803		JSR	PUTC	
8B11	E6	C4		LDB	,U	
8B13	CB	40		ADDB	#\$40	
8B15	BD	E803		JSR	PUTC	
8B18	E6	41		LDB	,U	
8B1A	CB	40		ADDB	#\$40	
8B1C	BD	E803		JSR	PUTC	
8B1F	37	06		PULU	A,B	
8B21	39			RTS		
8B22	36	36	RPLGR	PSHU	X,Y,A,B	Sauvegarde l'état des registres internes du 6809
8B24	10EE	8907		LDY	DIAG0+2	On charge Yo
8B28	10BC	890B		CMPLY	DIAG0+6	Compare Yo et Y1
8B2C	23	0C		BLS	TRACE	Si plus petit ou égal on ne modifie rien
8B2E	BE	890B		LDX	DIAG0+6	Si plus grand on échange Yo et Y1
8B31	1E	12		EXG	X,Y	
8B33	BF	890B		STX	DIAG0+6	
8B36	10BF	8907		STY	DIAG0+2	
8B3A	BE	8905	TRACE	LDX	DIAG0	Puis Xo
8B3D	BD	E80F		JSR	PLOT	Ecrit un point de coordonnées Yo, Xo. Les registres PLOTX et PLOTY sont chargés avec Xo et Yo
8B40	BE	8909		LDX	DIAG0+4	Charge X1
8B43	BD	E80C		JSR	DRAW	Trace un trait entre Xo, Yn et X1, Yn
8B46	10BC	890B		CMPLY	DIAG0+6	Sommes-nous à la fin du tracé
8B4A	27	04		BEQ	ZE	Oui. Alors on saute à la fin
8B4C	31	21		LEAY	,Y	Non. On incrémente Y de 1 pour tracer le trait suivant
8B4E	20	EA		BRA	TRACE	
8B50	39		ZE	RTS		
8B51	36	16	ALER	PSHU	X,B,A	Sauvegarde X,B,A
8B53	BE	6027		LDX	TIMEPT	Sauve l'état courant du pointeur utilisateur dans la pile U
8B56	36	10		PSHU	X	
8B58	0E	8B80		LDX	#INTER	Charge l'adresse du s/p d'interruption dans le pointeur
8B5B	BF	6027		STX	TIMEPT	
8B5E	C6	01		LDB	#\$01	Positionne l'indicateur
8B60	F7	8904		STB	INDIC	
8B63	C6	20		LDB	#\$20	Force le bit 5 du STATUS à 1
8B65	FA	6019		ORB	STATUS	
8B68	F7	6019		STB	STATUS	
8B6B	C6	09		LDB	#LIHY-LIBY-6	Initialise le compteur Y
8B6D	F7	8900		STB	PBY	
8B70	86	1C		LDA	#LIHX-LIBX	Initialise le compteur X
8B72	B7	8901		STA	PBX	
8B75	7D	8904	AR0	TST	INDIC	Teste l'indicateur pour savoir si une interruption a déjà eu lieu
8B78	27	16		BEQ	SORT	Oui. Alors on sort
8B7A	7A	8900		DEC	PBY	Non. On décrémente le compteur Y
8B7D	26	05		BNE	AZ0	Pas zéro, alors on saute à l'étiquette AZ0 pour continuer



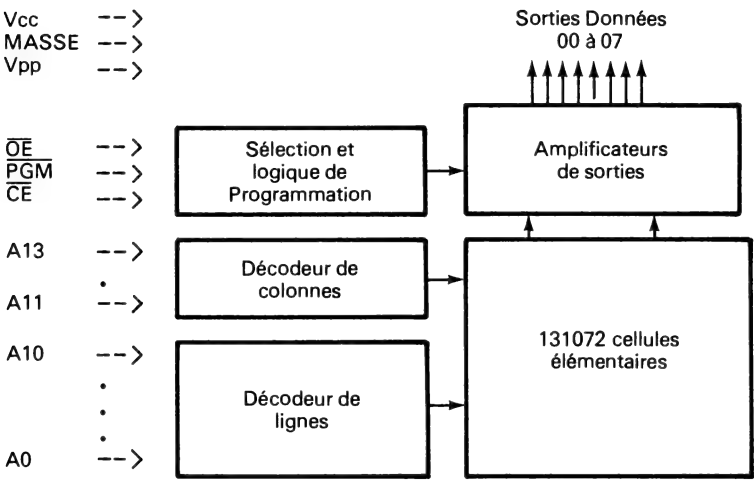
PROGRAMMATION DE MÉMOIRE EPROM

BUT : Utiliser les PIA (6821) en réalisant un programmeur d'EPROM.

Certains d'entre vous seront intéressés par la réalisation d'un programmeur de mémoire EPROM. Nous vous proposons de programmer des mémoires EPROM de relativement forte capacité. La 27128 ou la 2764 sont respectivement des mémoires de 16 K mots de 8 bits et 8 K mots de 8 bits. Soit 128 K bits pour la 27128 et 64 K bits pour la 2764 (K = 1024).

Ces mémoires vous permettront de sauvegarder, dans un même boîtier, une application relativement importante.

Organisation de l'EPROM 27128 :



Configuration et compatibilité :

27256	27128	2764	2732	2716	Bornes	2716	2732	2764	27128	2756
Vpp	Vpp	Vpp			1 28			Vcc	Vcc	Vcc
A12	A12	A12			2 27			PGM	PGM	A14
A7	A7	A7	A7	A7	3 26	Vcc	Vcc	N.C.	A13	A13
A6	A6	A6	A6	A6	4 25	A8	A8	A8	A8	A8
A5	A5	A5	A5	A5	5 24	A9	A9	A9	A9	A9
A4	A4	A4	A4	A4	6 23	Vpp	A11	A11	A11	A11
A3	A3	A3	A3	A3	7 23	OE	OE/Vpp	OE	OE	OE
A2	A2	A2	A2	A2	8 21	A10	A10	A10	A10	A10
A1	A1	A1	A1	A1	9 20	CE	CE	CE	CE	CE
A0	A0	A0	A0	A0	10 19	O7	O7	O7	O7	O7
O0	O0	O0	O0	O0	11 19	O6	O6	O6	O6	O6
O1	O1	O1	O1	O1	12 17	O5	O5	O5	O5	O5
O2	O2	O2	O2	O2	13 16	O4	O4	O4	O4	O4
GND	GND	GND	GND	GND	14 15	O3	O3	O3	O3	

N.C. : non connecté

On utilisera des PIA pour contrôler les entrées/sorties de la mémoire en programmation.

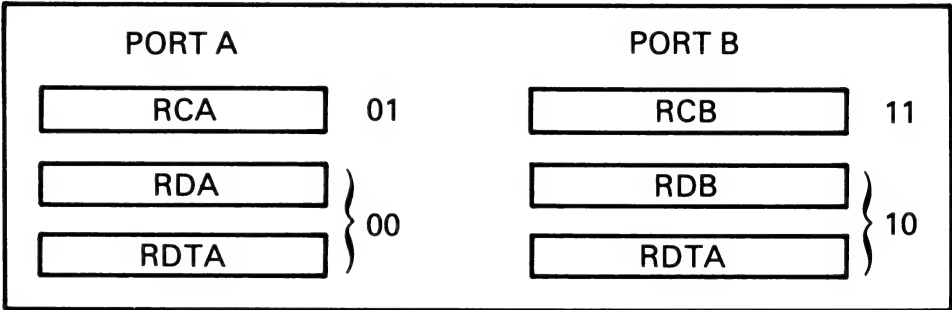
Les PIA 6821 :

Les PIA (*Peripheral Interface Adapter*) sont des circuits qui permettent de réaliser des liaisons, en mode parallèle, entre le microprocesseur et l'extérieur. Ces liaisons peuvent être bi-directionnelles, c'est-à-dire de l'extérieur vers le microprocesseur ou du micro vers l'extérieur.

Les PIA communiquent avec le 6809 à travers le bus de données.

Les PIA sont organisés en deux ports symétriques de 8 bits. (Port A et port B). Chaque port possède 3 registres qui sont accessibles par le 6809.

- Un registre de contrôle (RC) qui permet au 6809 de contrôler les lignes d'interruption C1 et C2 et d'accéder aux 2 autres registres.
- Un registre de données (RD) qui permet au microprocesseur de lire ou écrire les données.
- Un registre de direction (RDT) qui définit le sens de transfert.



Les registres de contrôle ou les couples registre de direction/données sont accessibles par le codage des entrées RS0 et RS1.

RS0	RS1	Registre concerné
0	0	RDA ou RDTA
1	0	RCA
0	1	RDB ou RDTB
1	1	RCB

Les bits 2 des registres de contrôle permettent de distinguer les registres de direction des registres de données.

Bit 2 de RC	Registre
1	Données
0	Direction

Le PIA dispose de 3 entrées de sélection (CS = *Chip Select*) ; pour qu'il soit *actif* il faut avoir la combinaison :

$CS0 = 1.CS1 = 1 \text{ et } \overline{CS2} = 0$

Mode de fonctionnement :

MODES \ Broches	\overline{CE}	\overline{OE}	\overline{PGM}	Vpp	Vcc	SORTIES
LECTURE	0	0	1	5V	5V	Données
ATTENTE	1	0	1	5V	5V	H.Z.
VERIFICATION	0	0	1	*12,5V	6V	Données
PRE.PROGRAM	0	1	1	*12,5V	6V	H.Z.
PROGRAMMATION	0	1	0	*12,5V	6V	Entrée

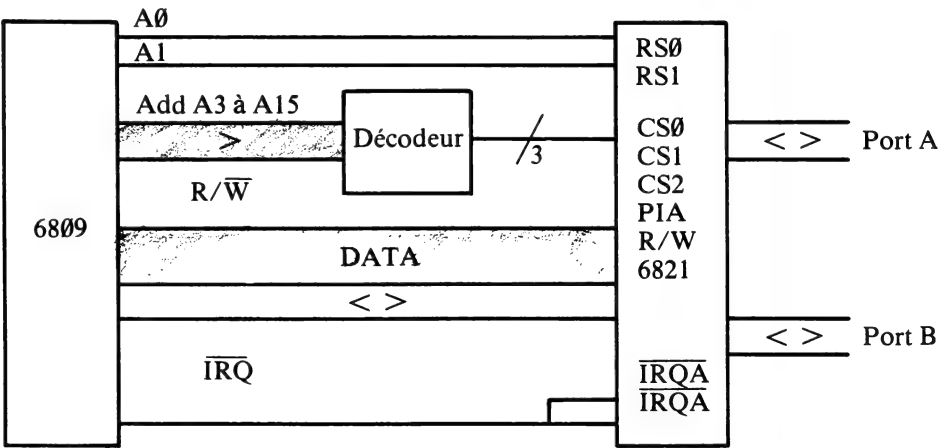
* Pour la 27128A, 21V pour la 27128.
H.Z. : Haute Impédance

L'état des bits des registres de direction permet de définir le sens du transfert sur les bits correspondants des registres de données.

Etat du bit	Sens de transfert
1	PIA → ext (sortie)
0	ext → PIA (entrée)

Pour adresser un PIA il convient donc d'appliquer les adresses A0 et A1 sur les entrées RS0 et RS1 et de decoder les autres bits d'adresses en utilisant éventuellement les entrées de sélection du boîtier.

Schéma minimum



Le programmeur

Pour programmer une EPROM il faut lui appliquer :

- les adresses
- les données
- des signaux de contrôle

Les adresses :

La 27128 a une capacité de 16 K mots. Il faut donc être en mesure de lui appliquer les codes d'adresses compris entre 0000 et 3FFF. Pour réaliser cette tâche nous utiliserons le PIA n° 1 qui est décodé aux adresses \$C004 à \$C007. Les poids faibles sur le port A et les poids forts sur le port B.

Comme les adresses sont issues d'un bus uni-directionnel, il conviendra donc de configurer le PIA n° 1 uniquement en sortie. Cette fonction est réalisée par les parties de programme comprises entre les adresses \$70CC et \$70F1 pour la séquence de programmation et de \$723D à \$7264 pour les séquences lecture et comparaison.

Les données :

La 27128 est une mémoire organisée en mots de 8 bits. Au cours de l'opération de programmation, les données peuvent être appliquées ou lues aux bornes de la mémoire. Pour réaliser cette tâche, nous utiliserons le port A du PIA n° 2 qui sera programmé en entrée ou en sortie en fonction de la tâche demandée. Les sous-programmes DATE et DATS assurent ces fonctions.

Les signaux de contrôle :

Les signaux de contrôle sont au nombre de cinq et sont délivrés par le port B du PIA n° 2. Port qui est programmé constamment en sortie, puisque les signaux de contrôle sont appliqués à la mémoire :

- \overline{CE} : permet de valider le boîtier de la mémoire
 $\overline{CE} = 0 \rightarrow$ boîtier valide
- \overline{OE} : permet la lecture des données (si $\overline{CE} = 0$)
 $\overline{OE} = 0 \rightarrow$ sorties des données
- \overline{PGM} : signal de programmation. Un niveau 0 sur cette entrée active la programmation
- V_{pp} : en phase de programmation, cette entrée doit être forcée à 21 V ou 12,5 V pour les mémoires de type A.
- V_{cc} : borne d'alimentation du circuit. En programmation V_{cc} doit être forcée à 6v.

Note sur le listing

Le listing donné ci-dessous a été obtenu en utilisant un lecteur de disquettes et en appelant le programme PROGAM.SCM par la directive INCLUDE.

Soit :

```
ORG $6C00
INCLUDE PROGRAM.SCM
```

Dans ce cas le programme objet est directement exécutable sur TO7 ou TO7-70 et laisse 16 K octets de RAM disponibles, entre 8000 et SFFF, utilisables comme *buffer* (mémoire tampon).

Si vous ne possédez qu'un lecteur de K7, vous ne pourrez pas procéder ainsi car la directive INCLUDE n'est pas autorisée avec un L.E.P.

Il vous faudra pratiquer en deux temps pour implanter votre programme binaire en mémoire.

1^{er} temps :

Assemblez le programme source en implantant le programme objet sur une K7 aux adresses spécifiées par la directive ORG

Soit la commande :

```
>AC:PROGRAM.BIN
```

2^e temps :

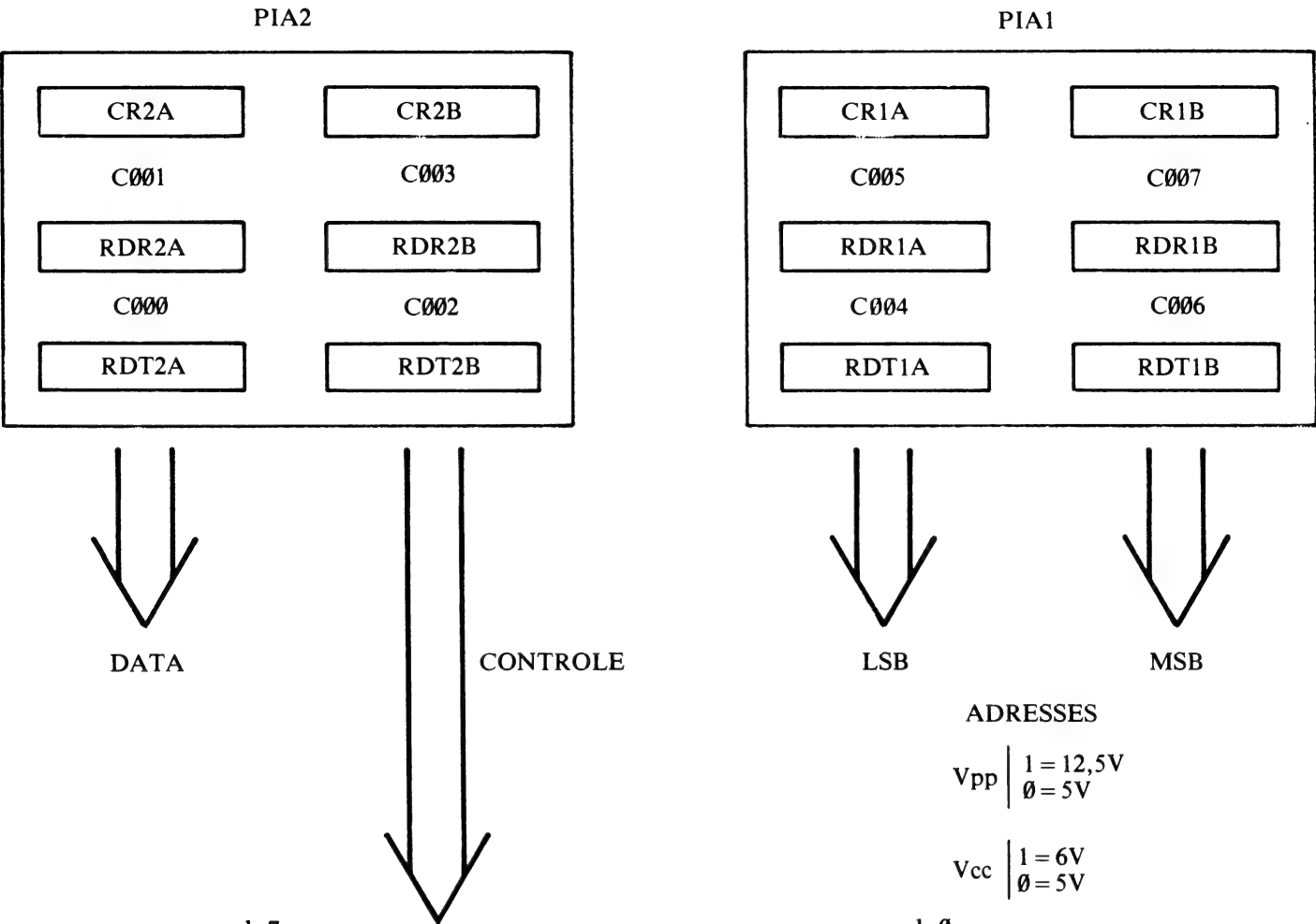
Implantez votre programme objet en mémoire, sous contrôle du moniteur.

Soit

```
# L C: PROGRAM.BIN
```

Puis lancez l'exécution par

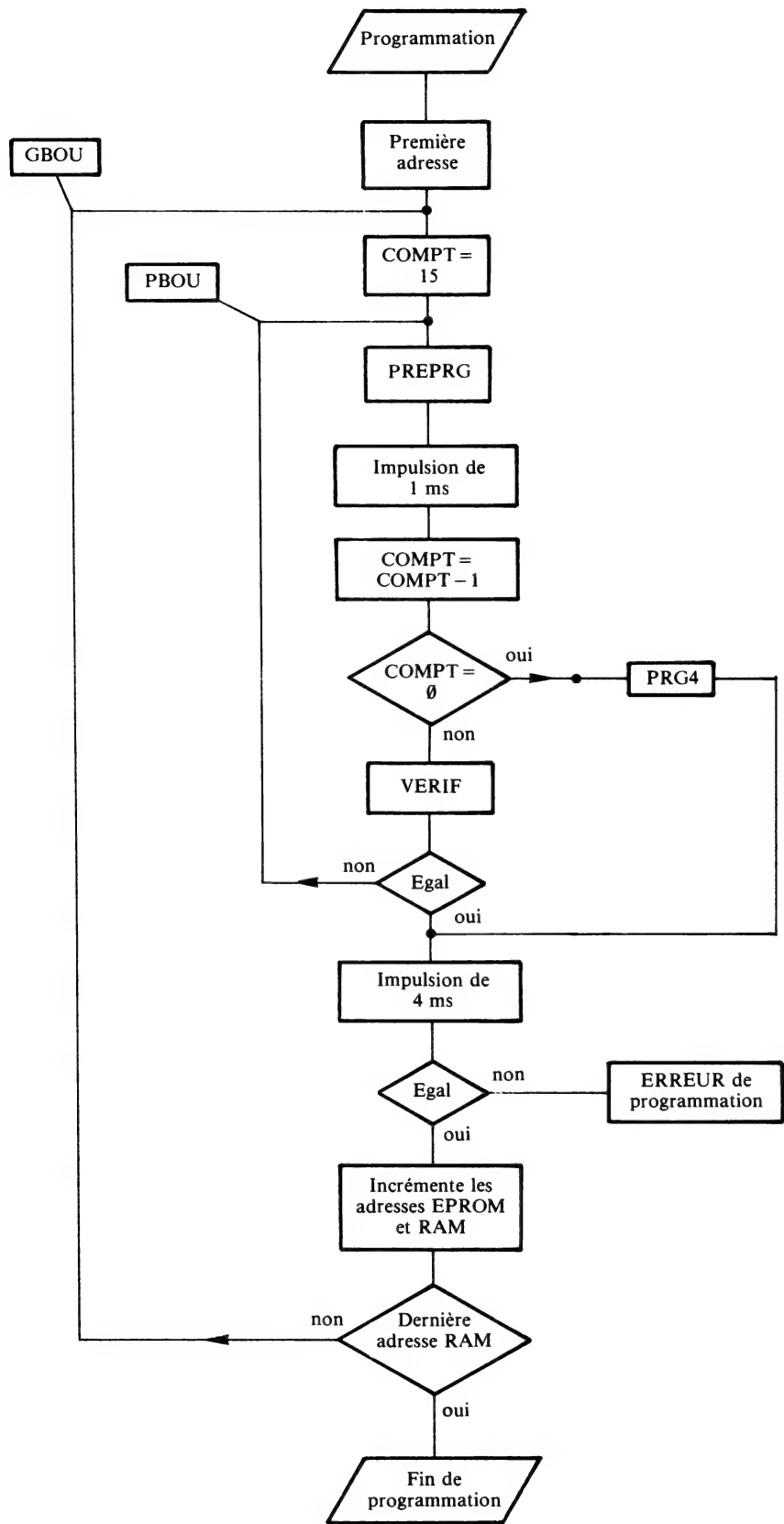
```
# GDEBUT
```



	b 7								b 0	
Modes	X	X	X	V c c	V p p	P G M	O E	C E		
Lecture	0	0	0	0	0	1	0	0	LECT	EQU \$04
Attente	0	0	0	0	0	1	0	1	ATTEN	EQU \$05
Vérification	0	0	0	1	1	1	0	0	VERIF	EQU \$1C
Pré-Program	0	0	0	1	1	1	1	0	PREPRG	EQU \$1E
Programmation	0	0	0	1	1	0	1	0	PRG	EQU \$1A

Adresses et codes utilisés dans le programme.

Organigramme de programmation :



* BG *** PROGRAMMATEUR D'EPROM *** RW *
* 27128A *

6000			ORG	\$6000	
6000			INCLUD	PROGRAMM.SCM	Non autorisée en version k7
			PNB	1	ComPteur
	0000		RDT2A	EQU	\$0000 Data Port 2A
	0000		RDR2A	EQU	\$0000 Direction Port 2A
	0001		CR2A	EQU	\$0001 Controle Port 2A
	0002		RDT2B	EQU	\$0002 Data Port 2B
	0002		RDR2B	EQU	\$0002 Direction Port 2B
	0003		CR2B	EQU	\$0003 Controle Port 2B
	0004		RDT1A	EQU	\$0004 Data Port 1A
	0004		RDR1A	EQU	\$0004 Direction Port 1A
	0005		CR1A	EQU	\$0005 Controle Port 2B
	0006		RDT1B	EQU	\$0006 Data Port 1B
	0006		RDR1B	EQU	\$0006 Direction Port 1B
	0007		CR1B	EQU	\$0007 Controle Port 1B
	0004		LECT	EQU	\$04 % 00000100
	0010		VERIF	EQU	\$10 % 00011100
	0005		ATTEN	EQU	\$05 % 00000101
	001E		PREPRG	EQU	\$1E % 00011110
	001A		PRG	EQU	\$1A % 00011010
	0000		CR	EQU	\$00
	0016		NVAL	EQU	\$16
6001	30	31 32 33	VAL	FCB	'0','1','2','3','4','5','6','7','8
6005	34	35 36 37			
6009	38				
600A	39	41 42 43		FCB	'9','A','B','C','D','E','F
600E	44	45 46			
	0000		ADD0	EQU	\$0000
	3FFF		ADDM	EQU	\$3FFF \$1FFF Pour 2764
6011	38	10 0E FF	CARUT	FCB	\$38,\$10,\$0E,\$FF,\$FF,\$0E
6015	FF	0E			
6017	10	38 10 38		FCB	\$10,\$38,\$10,\$38,\$70,\$FF
601B	70	FF			
601D	FF	70 38 10		FCB	\$FF,\$70,\$38,\$10
6021	1F	53 61 1B	FLED	FCB	\$1F,\$53,\$61,\$1B,\$41,\$20
6025	41	80			
6027	00			FCB	\$00
6028	1F	53 60 1B	FLEG	FCB	\$1F,\$53,\$60,\$1B,\$41,\$81
602C	41	81			
602E	00			FCB	\$00
602F	1F	53 60 1B	FLOG	FCB	\$1F,\$53,\$60,\$1B,\$41,\$81
6033	41	81			
6035	80	00		FCB	\$80,\$00
6037	1F	12 14 1F	MG0	FCB	\$1F,\$12,\$14,\$1F,\$20,\$20
603B	20	20			
603D	00			FCB	\$00
603E	1F	41 41 1B		FCB	\$1F,\$41,\$41,\$1B,\$43,\$1B
6042	43	1B			
6044	40	1B 62		FCB	\$40,\$1B,\$62
6047	20	20 20 20		FCC	/ PROGRAMMATEU/
604B	20	20 20 20			
604F	20	20 20 50			
6053	52	4F 47 52			
6057	41	40 40 41			
605B	54	45 55			

605E 52 20 44 27	FCC	/9-D'EPROM	
6062 45 50 52 4F			
6066 40 20 20 20			
606A 20 20 20 20			
606E 1B 40 1F 44	FCB	\$1B,\$40,\$1F,\$44,\$41,\$1B	
6072 41 1B			
6074 41	FCB	\$41	
6075 20 20 20 20	FCC	/	27128
6079 20 20 20 20			
607D 20 20 20 20			
6081 20 20 20 20			
6085 20 32 37 31			
6089 32 38 20			
608C 1B 40	FCB	\$1B,\$40	
608E 1F 4A 45 1B	FCB	\$1F,\$4A,\$45,\$1B,\$45	
6092 45			
6093 31 20 40 45	FCC	/1-LECTURE D'EPROM/	
6097 43 54 55 52			
609B 45 20 44 27			
609F 45 50 52 4F			
60A3 40			
60A4 1F 40 45 1B	FCB	\$1F,\$40,\$45,\$1B,\$45	
60A8 45			
60A9 32 20 50 52	FCC	/2-PROGRAMMATION/	
60AD 4F 47 52 41			
60B1 40 40 41 54			
60B5 49 4F 4E			
60B8 1F 4E 45 1B	FCB	\$1F,\$4E,\$45,\$1B,\$45	
60BC 45			
60BD 33 20 43 4F	FCC	/3-COMPARAISON/	
60C1 40 50 41 52			
60C5 41 49 53 4F			
60C9 4E			
60CA 1F 57 41 1B	FCB	\$1F,\$57,\$41,\$1B,\$44	
60CE 44			
60CF 54 41 50 45	FCC	/TAPEZ VOTRE CHOIX AU CL/	
60D3 5A 20 56 4F			
60D7 54 52 45 20			
60DB 43 48 4F 49			
60DF 58 20 41 55			
60E3 20 43 40			
60E6 41 56 49 45	FCC	/AVIER	
60EA 52 20 20 20			
60EE 20 20 20 20			
60F2 20 20 20 20			
60F6 20			
60F7 1F 50 5C 1B	FCB	\$1F,\$50,\$5C,\$1B,\$42	
60FB 42			
60FC 20 52 41 40	FCC	RAM EPROM	
6D00 20 20 45 50			
6D04 52 4F 40 20			
6D08 1F 52 56 1B	FCB	\$1F,\$52,\$56,\$1B,\$46	
6D0C 46			
6D0D 44 45 42 55	FCC	/DEBUT/	
6D11 54			
6D12 1F 54 56	FCB	\$1F,\$54,\$56	
6D15 46 49 4E	FCC	/FIN/	
6D18 00	FCB	\$00	
6D19 1F 57 41 1B MG1	FCB	\$1F,\$57,\$41,\$1B,\$44,\$07	
6D1D 44 07			

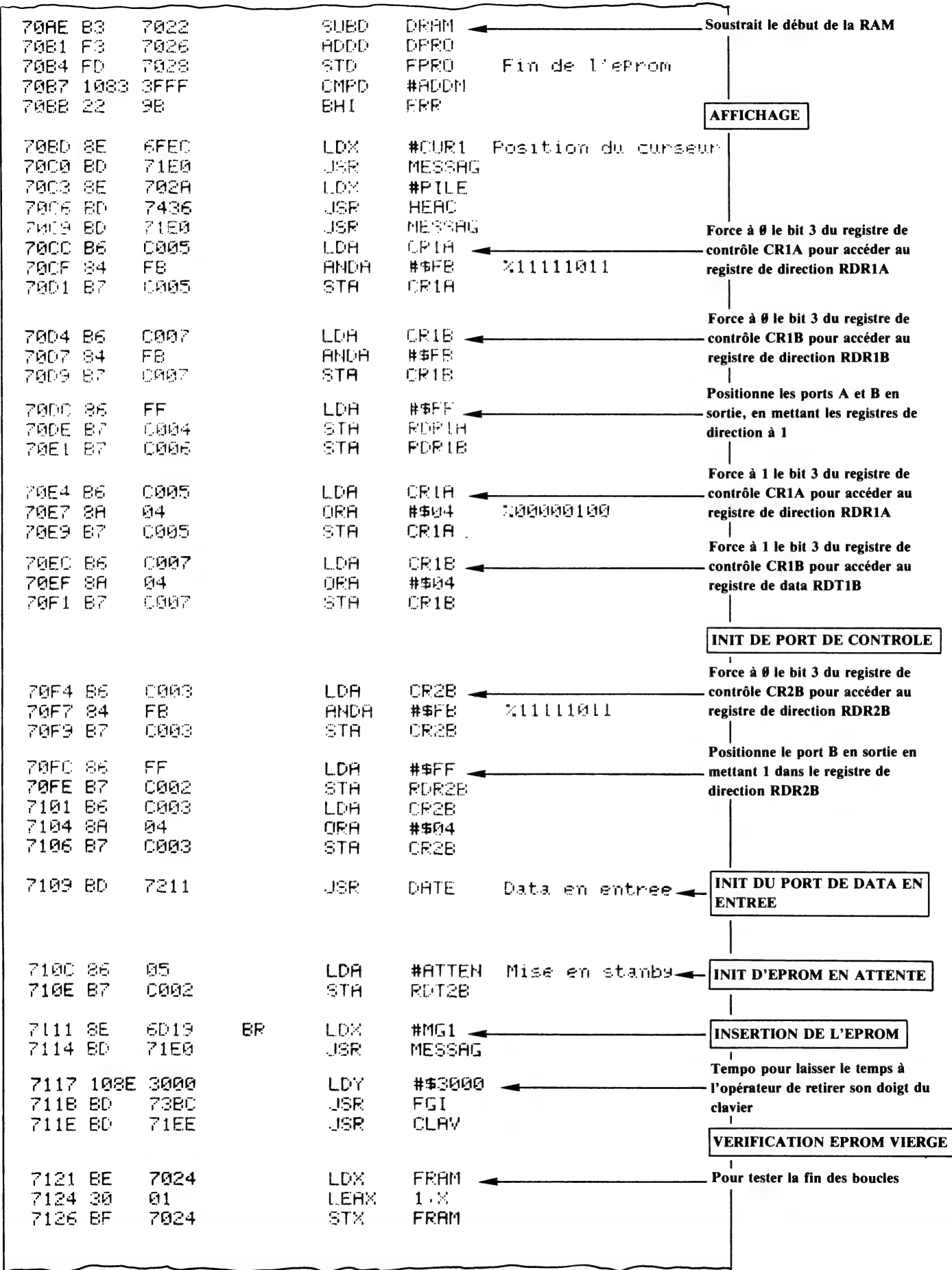
6D1F 49 4E 53 45		FCC	/INSEPEZ L'EPROM ET PRES/
6D23 52 45 5A 20			
6D27 40 27 45 50			
6D2B 52 4F 4D 20			
6D2F 45 54 20 50			
6D33 52 45 53			
6D36 53 45 5A 20		FCC	/SEZ UNE TOUCHE
6D3A 55 4E 45 20			
6D3E 54 4F 55 43			
6D42 48 45 20 20			
6D46 20			
6D47 00		FCB	\$00
6D48 1F 57 41 1B MG2		FCB	\$1F,\$57,\$41,\$1B,\$44,\$07
6D4C 44 07			
6D4E 40 45 4D 4F		FCC	/MEMOIRE VIERGE
6D52 49 52 45 20			
6D56 56 49 45 52			
6D5A 47 45 20 20			
6D5E 20 20 20 20			
6D62 20 20 20			
6D65 20 20 20 20		FCC	/
6D69 20 20 20 20			
6D6D 20 20 20 20			
6D71 20 20 20 20			
6D75 20			
6D76 00		FCB	\$00
6D77 1F 57 41 1B MG3		FCB	\$1F,\$57,\$41,\$1B,\$41,\$1B
6D7B 41 1B			
6D7D 61 07		FCB	\$61,\$07
6D7F 45 52 52 45		FCC	/ERREUR !! MEMOIRE PAS VI/
6D83 55 52 20 21			
6D87 21 4D 45 4D			
6D8B 4F 49 52 45			
6D8F 20 50 41 53			
6D93 20 56 49			
6D96 45 52 47 45		FCC	/ERGE !! /
6D9A 20 21 21 20			
6D9E 20 20 20 20			
6DA2 20 20 20 20			
6DA6 20			
6DA7 00		FCB	\$00
6DA8 1F 57 41 1B MG4		FCB	\$1F,\$57,\$41,\$1B,\$41,\$1B
6DAC 41 1B			
6DAE 61 07		FCB	\$61,\$07
6DB0 45 52 52 45		FCC	/ERREUR !! PAS DE PROGRA/
6DB4 55 52 20 21			
6DB8 21 50 41 53			
6DBC 20 44 45 20			
6DC0 50 52 4F 47			
6DC4 52 41			
6DC6 4D 4D 41 54		FCC	/MMATION
6DCA 49 4F 4E 20			
6DCE 20 20 20 20			
6DD2 20 20 20 20			
6DD6 20			
6DD7 00		FCB	\$00
6DD8 1F 57 41 1B MG5		FCB	\$1F,\$57,\$41,\$1B,\$44,\$07
6DDC 44 07			
6DDE 4D 45 4D 4F		FCC	/MEMOIRE PROGRAMMEE
6DE2 49 52 45 20			

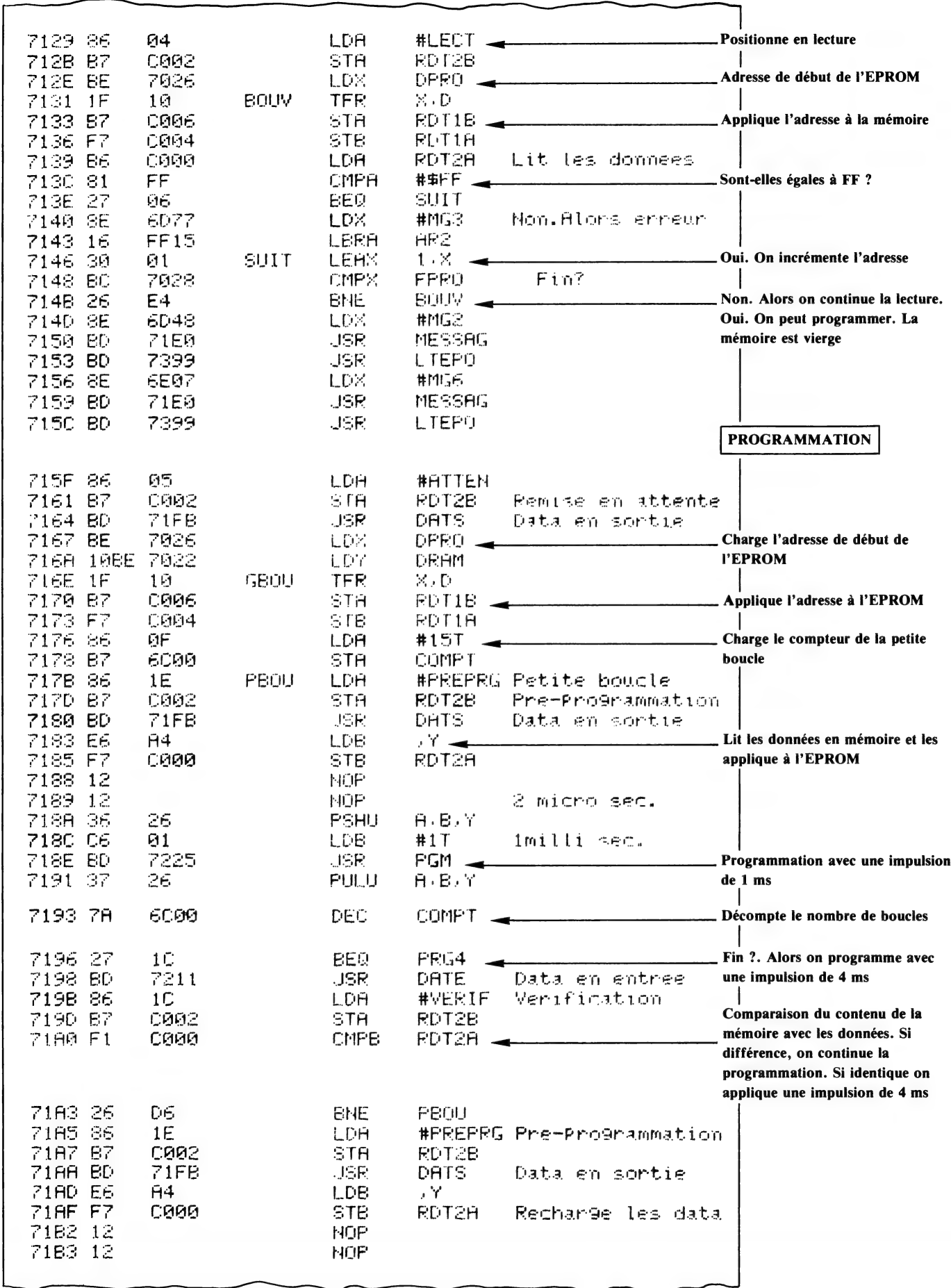
6DE6	50	52	4F	47			
6DEA	52	41	40	40			
6DEE	45	45	20	20			
6DF2	20	20	20				
6DF5	20	20	20	20	F00		
6DF9	20	20	20	20			
6DFD	20	20	20	20			
6E01	20	20	20	20			
6E05	20						
6E06	00				FCB	\$00	
6E07	1F	57	41	1B	MG6	FCB	\$1F,\$57,\$41,\$1B,\$44,\$07
6E08	44	07					
6E0D	50	52	4F	47	F00		/PROGRAMMATION EN COURS
6E11	52	41	40	40			
6E15	41	54	49	4F			
6E19	4E	20	45	4E			
6E1D	20	43	4F	55			
6E21	52	53	20				
6E24	20	20	20	20	F00		
6E28	20	20	20	20			
6E2C	20	20	20	20			
6E30	20	20	20	20			
6E34	20						
6E35	00				FCB	\$00	
6E36	1F	57	41	1B	MG7	FCB	\$1F,\$57,\$41,\$1B,\$44,\$07
6E3A	44	07					
6E3C	4C	45	43	54	F00		/LECTURE TERMINEE /
6E40	55	52	45	20			
6E44	54	45	52	4D			
6E48	49	4E	45	45			
6E4C	20	20	20	20			
6E50	20	20	20				
6E53	20	20	20	20	F00		
6E57	20	20	20	20			
6E5B	20	20	20	20			
6E5F	20	20	20	20			
6E63	20						
6E64	00				FCB	\$00	
6E65	1F	57	41	1B	MG9	FCB	\$1F,\$57,\$41,\$1B,\$44,\$07
6E69	44	07					
6E6B	43	4F	4D	50	F00		/COMPARAISON TERMINEE
6E6F	41	52	41	49			
6E73	53	4F	4E	20			
6E77	54	45	52	4D			
6E7B	49	4E	45	45			
6E7F	20	20	20				
6E82	20	20	20	20	F00	/	/
6E86	20	20	20	20			
6E8A	20	20	20	20			
6E8E	20	20	20	20			
6E92	20						
6E93	00				FCB	\$00	
6E94	1F	57	41	1B	MG8	FCB	\$1F,\$57,\$41,\$1B,\$41,\$1B
6E98	41	1B					
6E9A	61	07			FCB	\$61,\$07	
6E9C	45	52	52	45	F00		/ERREUR !! RAM ET EPROM /
6EA0	55	52	20	21			
6EA4	21	52	41	4D			
6EA8	20	45	54	20			
6EAC	45	50	52	4F			

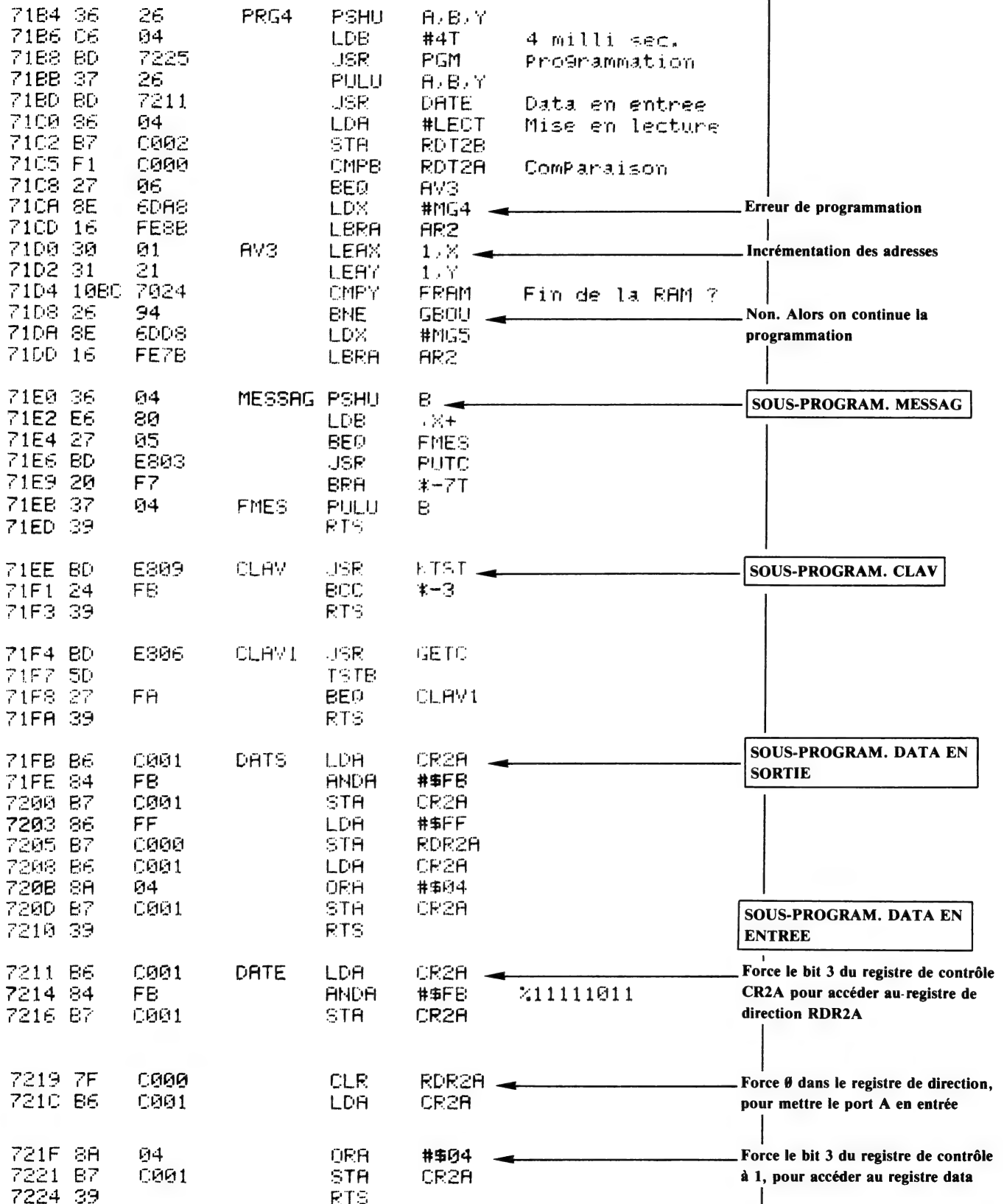
6EB0	40	20			
6EB2	44	49	46	46	FCC /DIFFERENTES
6EB6	45	52	45	4E	
6EBA	54	45	53	20	
6EBE	20	20	20	20	
6EC2	20				
6EC3	00				FCB \$00
6EC4	1F	57	41	1B	MG10 FCB \$1F,\$57,\$41,\$1B,\$44,\$07
6EC8	44	07			
6ECA	40	45	43	54	FCC /LECTURE EN COURS
6ECE	55	52	45	20	
6ED2	45	4E	20	43	
6ED6	4F	55	52	53	
6EDA	20	20	20	20	
6EDE	20	20	20		
6EE1	20	20	20	20	FCC
6EE5	20	20	20	20	
6EE9	20	20	20	20	
6EED	20	20	20	20	
6EF1	20				
6EF2	00				FCB \$00
6EF3	1F	57	41	1B	MG11 FCB \$1F,\$57,\$41,\$1B,\$44,\$07
6EF7	44	07			
6EF9	43	4F	4D	50	FCC /COMPARAISON EN COURS
6EFD	41	52	41	49	
6F01	53	4F	4E	20	
6F05	45	4E	20	43	
6F09	4F	55	52	53	
6F0D	20	20	20		
6F10	20	20	20	20	FCC
6F14	20	20	20	20	
6F18	20	20	20	20	
6F1C	20	20	20	20	
6F20	20				
6F21	00				FCB \$00
6F22	1F	57	41	1B	MG12 FCB \$1F,\$57,\$41,\$1B,\$44,\$07
6F26	44	07			
6F28	45	4E	54	52	FCC /ENTREZ L'ADRESSE DE DEB/
6F2C	45	5A	20	4C	
6F30	27	41	44	52	
6F34	45	53	53	45	
6F38	20	44	45	20	
6F3C	44	45	42		
6F3F	55	54	20	44	FCC /UT DE LA RAM /
6F43	45	20	4C	41	
6F47	20	52	41	4D	
6F4B	20	20	20	20	
6F4F	20				
6F50	1F	52	5C	1B	CUR4 FCB \$1F,\$52,\$5C,\$1B,\$41
6F54	41				
6F55	00				FCB \$00
6F56	1F	57	41	1B	MG13 FCB \$1F,\$57,\$41,\$1B,\$44,\$07
6F5A	44	07			
6F5C	45	4E	54	52	FCC /ENTREZ L'ADRESSE DE DEB/
6F60	45	5A	20	4C	
6F64	27	41	44	52	
6F68	45	53	53	45	
6F6C	20	44	45	20	
6F70	44	45	42		
6F73	55	54	20	44	FCC /UT DE L'EPROM

6F77	45	20	40	27				
6F7B	45	50	52	4F				
6F7F	40	20	20	20				
6F83	20							
6F84	1F	52	62	1B	CUR3	FCB	\$1F,\$52,\$62,\$1B,\$43	
6F88	43							
6F89	00					FCB	\$00	
6F8A	1F	57	41	1B	MG14	FCB	\$1F,\$57,\$41,\$1B,\$44,\$07	
6F8E	44	07						
6F90	45	4E	54	52		FCC	/ENTREZ L'ADRESSE DE FIN/	
6F94	45	5A	20	40				
6F98	27	41	44	52				
6F9C	45	53	53	45				
6FA0	20	44	45	20				
6FA4	46	49	4E					
6FA7	20	44	45	20		FCC	DE LA RAM	
6FAB	40	41	20	52				
6FAF	41	40	20	20				
6FB3	20	20	20	20				
6FB7	20							
6FB8	1F	54	50	1B	CUR2	FCB	\$1F,\$54,\$50,\$1B,\$41	
6FBC	41							
6FBD	00					FCB	\$00	
6FBE	1F	57	41	1B	MG15	FCB	\$1F,\$57,\$41,\$1B,\$44,\$07	
6FC2	44	07						
6FC4	45	4E	54	52		FCC	/ENTREZ L'ADRESSE DE FIN/	
6FC8	45	5A	20	40				
6FCC	27	41	44	52				
6FD0	45	53	53	45				
6FD4	20	44	45	20				
6FD8	46	49	4E					
6FDB	20	44	45	20		FCC	/ DE L'EPROM	
6FDF	40	27	45	50				
6FE3	52	4F	40	20				
6FE7	20	20	20	20				
6FEB	20							
6FEC	1F	54	62	1B	CUR1	FCB	\$1F,\$54,\$62,\$1B,\$43	
6FF0	43							
6FF1	00					FCB	\$00	
6FF2	1F	57	41	1B	NGER	FCB	\$1F,\$57,\$41,\$1B,\$41,\$1B	
6FF6	41	1B						
6FF8	61	07				FCB	\$61,\$07	
6FFA	45	52	52	45		FCC	/ERREUR !!	
6FFE	55	52	20	21				
7002	21	20	20	20				
7006	20	20	20	20				
700A	20	20	20	20				
700E	20	20						
7010	20	20	20	20		FCC		
7014	20	20	20	20				
7018	20	20	20	20				
701C	20	20	20	20				
7020	20							
7021	00					FCB	\$00	
7022		DRAM				RMB	2	Debut RAM
7024		FRAM				RMB	2	Fin RAM
7026		DPRO				RMB	2	Debut EPROM
7028		FPRO				RMB	2	Fin EPROM
702A		PILE				RMB	4	Zone de stockage
702E		DELI				RMB	1	Delimiteur

702F			CORPAR	RMB	1	Compteur de frappes	
7030			INDIC	RMB	1	←	1 = lecture 0 = comparaison
	E803		PUTC	EOU	\$E803	Ecran	
	E809		KTST	EOU	\$E809	Clavier	
	E806		GETC	EOU	\$E806	Clavier	
	E703		PRL	EOU	\$E703	←	Registre de donnée (système)
INIT DES PORTS D'ADRESSES EN SORTIE							
7031	0E	7FFF	DEBUT	LNU	#7FFF	←	Pointeur U. La zone réservée à la pile doit en dehors de la RAM utilisée pour le transfert des données
7034	7F	7030	AR1	CLR	INDIC	←	R.a.Z. de l'indicateur
7037	8E	6C37		LDX	#MG0	←	Pointe sur le message 0
703A	BD	71E0		JSR	MESSAG		Entre un caractère à partir du clavier
703D	BD	71F4		JSR	CLAVI	←	
7040	C1	31		CMPB	#'1	Egal a un ?	
7043	1026	0006		LBNE	*+\$0A	←	Non. Alors on poursuit la comparaison
7046	7C	7030		INC	INDIC	←	Oui. Alors on positionne l'indicateur à 1
7049	16	01F1		LBRA	AV1		
704C	C1	32		CMPB	#'2	Egal a deux?	
704E	1027	0011		LBEO	AV2	←	Oui. Alors on saute en programmation
7052	C1	33		CMPB	#'3	Egal a trois	
7054	1027	01E5		LBEO	AV1	←	Oui. Alors on saute au tronçon de programme commun à la lecture et la comparaison
7058	8E	6FF2	ERR	LDX	#MGER	Non.Alors erreur	
705B	BD	71E0	AR2	JSR	MESSAG		
705E	BD	7399		JSR	LTEPO	Longue tempo	
7061	D1			BRA	AR1		
7063	8E	6F22	AV2	LDX	#MG12	Debut de la RAM	
7066	BD	71E0		JSR	MESSAG		
7069	8E	6C11		LDX	#CARUT	←	Caractère utilisateur
706C	BF	602D		STX	\$602D		
706F	8E	6C21		LDX	#FLED		
7072	BD	71E0		JSR	MESSAG		
7075	8E	6F50		LDX	#CUR4		
7078	BD	71E0		JSR	MESSAG		
707B	86	04		LDA	#\$04	4 digits	
707D	BD	73C1		JSR	NOMHEX		
7080	FD	7022		STD	DRAM		
7083	8E	6F8A		LDX	#MG14	Fin de la RAM	
7086	BD	71E0		JSR	MESSAG		
7089	86	04		LDA	#\$04	4 digits	
708B	BD	73C1		JSR	NOMHEX		
708E	FD	7024		STD	FRAM		
7091	10B3	7022		CMPD	DRAM	←	Compare les adresses de la RAM (début <fin)
7095	23	C1		BLS	ERR	Erreur	
7097	8E	6F56		LDX	#MG13	Debut de l'EPROM	
709A	BD	71E0		JSR	MESSAG		
709D	86	04		LDA	#\$04	4 digits	
709F	BD	73C1		JSR	NOMHEX		
70A2	FD	7026		STD	DPRO		
70A5	10B3	3FFF		CMPD	#ADDN	←	Compare le début de l'EPROM avec sa capacité max
70A9	2A	AD		BHI	ERR		
70AB	FC	7024		LDD	FRAM	←	Calcule la fin de l'EPROM







7225	86	78	PGM	LDA	#120T	120=1ms	
7227	3D			MUL		Durée du Pulse	
7228	1A	10		ORCC	##10	Masque IRQ	
722A	1F	02		TFR	D,Y	Charge le "chrono"	
722C	86	1A		LDA	#PRG		Mise en programmation
722E	B7	C002		STA	RDT2B		
7231	31	3F	BT	LEAY	-1,Y		Décompte le temps d'une impulsion de programmation
7233	26	FC		BNE	BT		
7235	86	1E		LDA	#PREPRG		Remise en post ou pré-programmation
7237	B7	C002		STA	RDT2B		
723A	1C	EF		ANDCC	##EF	Demasque IRQ	
723C	39			RTS			
INIT PIA + MODE LECTURE							
723D	86	C007	AV1	LDA	CR1B	RDT1B -> SORTIE	
7240	84	FB		ANDR	##FB		
7242	B7	C007		STA	CR1B		
7245	86	FF		LDA	##FF		
7247	B7	C006		STA	RDT1B		
724A	86	C007		LDA	CR1B		
724D	8A	04		ORA	##04		
724F	B7	C007		STA	CR1B		
7252	86	C005		LDA	CR1A	RDT1A -> SORTIE	
7255	84	FB		ANDR	##FB		
7257	B7	C005		STA	CR1A		
725A	86	FF		LDA	##FF		
725C	B7	C004		STA	RDT1A		
725F	86	C005		LDA	CR1A		
7262	8A	04		ORA	##04		
7264	B7	C005		STA	CR1A		
7267	86	C001		LDA	CR2A	RDT2A <- ENTREE	
726A	84	FB		ANDR	##FB		
726C	B7	C001		STA	CR2A		
726F	7F	C000		CLP	RDT2A		
7272	8A	04		ORA	##04		
7274	B7	C001		STA	CR2A		
7277	86	C003		LDA	CR2B	RDT2B -> SORTIE	
727A	84	FB		ANDR	##FB		
727C	B7	C003		STA	CR2B		
727F	86	FF		LDA	##FF		
7281	B7	C002		STA	RDT2B		
7284	86	C003		LDA	CR2B		
7287	8A	04		ORA	##04		
7289	B7	C003		STA	CR2B		
728C	86	1C		LDA	##1C	Mode lecture	
728E	B7	C002		STA	RDT2B		
7291	8E	6F56		LDX	#MG13	Debut de l'eprom	
7294	8D	71E0		JSR	MESSAG		
7297	8E	6C11		LDX	#CARUT	Pointeur utilisateur	
729A	8F	602D		STX	\$602D		
729D	7D	7030		TST	INDIC		
72A0	27	0E		BEQ	RT2		
72A2	8E	6C28		LDX	#FLEG		
72A5	8D	71E0		JSR	MESSAG		
72A8	8E	6F84		LDX	#CUR3		
72AB	8D	71E0		JSR	MESSAG		
72AE	20	0C		BRA	RT1		
72B0	8E	6C2F	RT2	LDX	#FLDG		
72B3	8D	71E0		JSR	MESSAG		

72B6	8E	6F84		LDX	#CUR3	
72B9	BD	71E0		JSR	MESSAG	
72BC	86	04	RT1	LDA	#\$04	4 digits
72BE	BD	73C1		JSR	NOMHEX	
72C1	FD	7026		STD	DPRO	
72C4	10B3	3FFF		CMPO	#ADDM	
72C8	1022	FD8C		LBHI	ERR	
72CC	8E	6FBE		LDX	#MG15	Fin de l'ePROM
72CF	BD	71E0		JSR	MESSAG	
72D2	86	04		LDA	#\$04	4 digits
72D4	BD	73C1		JSR	NOMHEX	
72D7	FD	7028		STD	FPRO	
72DA	10B3	7026		CMPO	DPRO	Compare les adresses de l'EPROM (début < fin)
72DE	1023	FD76		LBLS	ERR	Erreur
72E2	10B3	3FFF		CMPO	#ADDM	Compare à l'adresse max de l'EPROM
72E6	1022	FD6E		LBHI	ERR	
72EA	8E	6F22		LDX	#MG12	Debut de la RAM
72ED	BD	71E0		JSR	MESSAG	
72F0	86	04		LDA	#\$04	4 digits
72F2	BD	73C1		JSR	NOMHEX	
72F5	FD	7022		STD	DRAM	
72F8	FD	7028		LDD	FPRO	Calcul la fin de la RAM
72FB	B3	7026		SUBD	DPRO	Soustrait le début de l'EPROM
72FE	F3	7022		ADD	DRAM	
7301	FD	7024		STD	FRAM	AFFICHAGE
7304	8E	6FB8		LDX	#CUR2	Position du curseur
7307	BD	71E0		JSR	MESSAG	
730A	8E	702A	ACA	LDX	#PILE	
730D	BD	7436		JSR	HEAC	
7310	BD	71E0		JSR	MESSAG	
7313	BE	7024		LDX	FRAM	Pour tester la fin
7316	30	01		LEAX	1,X	
7318	BF	7024		STX	FRAM	
731B	BE	7028		LDX	FPRO	
731E	30	01		LEAX	1,X	
7320	BF	7028		STX	FPRO	
7323	8E	6D19		LDX	#MG1	Insertion EPROM
7326	BD	71E0		JSR	MESSAG	
7329	10BE	3000		LDY	#\$3000	Tempo pour laisser le temps à l'opérateur de retirer son doigt du clavier
732D	BD	73BC		JSR	FGI	
7330	BD	71EE		JSR	CLAV	
7333	7D	7030		TST	INDIC	
7336	27	2C		BEQ	AV8	
7338	8E	6E04		LDX	#MG10	
733B	BD	71E0		JSR	MESSAG	
733E	BE	7026	BR0	LDX	DPRO	Adresses de départ
7341	10BE	7022		LDY	DRAM	
7345	1F	10	BOU	TFR	X,D	
7347	B7	C006		STA	RDT1B	APPLIQUE l'adresse
734A	F7	C004		STB	RDT1A	
734D	BD	73AF		JSR	TEMPO	
7350	B6	C000		LDA	RDT2A	Lecture data
7353	A7	A4		STA	Y	Stocke dans la RAM
7355	30	01		LEAX	1,X	Incrémente les adresses
7357	31	21		LEAY	1,Y	
7359	BC	7028		CMPL	FPRO	Fin?
735C	26	E7		BNE	BOU	Non. on continue

735E	8E	6E36		LDX	#MG7	
7361	16	FCF7		LBRA	AR2	
7364	8E	6EF3	AV8	LDX	#MG11	
7367	BD	71E0		JSR	MESSAG	
736A	BE	7026		LDX	DPRO	Adresses de départ
736D	10BE	7022		LDY	DRAM	
7371	1F	10	FOU	TFR	X:0	
7373	B7	C006		STA	RDT1B	APPLIQUE l'adresse
7376	F7	C004		STB	RDT1A	
7379	BD	73AF		JSR	TEMPO	
737C	A6	A4		LDA	.Y	Lit la RAM
737E	B1	C000		CMPL	RDT2A	← Comparaison avec les valeurs contenues dans l'EPROM
7381	27	06		BEQ	AV7	
7383	8E	6E94		LDX	#MG8	Erreur
7386	16	FC02		LBRA	AR2	
7389	30	01	AV7	LEAX	1.X	← Incrémente les adresses
738B	31	21		LEAY	1.Y	
738D	10BC	7024		CMPL	FRAM	Fin?
7391	26	DE		BNE	FOU	Non. On continue
7393	8E	6E65		LDX	#MG9	
7396	16	FC02		LBRA	AR2	
7399	108E	FFFF	LTEPO	LDY	#\$FFFF	
739D	BD	73BC		JSR	FGI	
73A0	108E	FFFF		LDY	#\$FFFF	
73A4	BD	73BC		JSR	FGI	
73A7	108E	FFFF		LDY	#\$FFFF	
73AB	BD	73BC		JSR	FGI	
73AE	39			RTS		
73AF	36	20	TEMPO	PSHU	Y	
73B1	108E	0050		LDY	#\$0050	
73B5	31	3F	FGO	LEAY	-1.Y	
73B7	26	FC		BNE	FGO	
73B9	37	20		PULU	Y	
73BB	39			RTS		
73BC	31	3F	FGI	LEAY	-1.Y	
73BE	26	FC		BNE	FGI	
73C0	39			RTS		
ENTREE D'UN NOMBRE HEXADECIMAL						
73C1	36	10	NOMHEX	PSHU	X	
73C3	F6	E7C3		LDB	PRC	Force en majuscules
73C6	C4	F7		ANDB	#\$F7	
73C8	F7	E7C3		STB	PRC	
73CB	8E	702A		LDX	#PILE	
73CE	B7	702F		STA	COFRAP	
73D1	BD	71F4	CONTI	JSR	CLAV1	
73D4	BD	7421		JSR	COMPAR	
73D7	26	F8		BNE	CONTI	
73D9	BD	E803		JSR	PUTC	
73DC	E7	80		STB	.X+	
73DE	7A	702F		DEC	COFRAP	
73E1	26	EE		BNE	CONTI	
73E3	8E	702A		LDX	#PILE	
73E6	BD	73EC		JSR	COASHE	
73E9	37	10		PULU	X	
73EB	39			RTS		

CONVERSION ASCII/HEXA

```

73EC 36 01      COASHE PSHU  CC
73EE E6 84      RA1    LDB   ,X
73F0 C0 30      SUBB   ##30
73F2 C1 09      CMPB   ##9
73F4 2F 02      BLE    *+4
73F6 C0 07      SUBB   ##7
73F8 E7 80      STB    ,X+
73FA 4A        DECA
73FB 26 F1      BNE    RA1
73FD 30 1F      LEAX   -1,X
73FF 36 10      PSHU   X
7401 8E 702A    LDX    #FILE
7404 E6 80      ETI    LDB   ,X+
7406 58        LSLB
7407 58        LSLB
7408 58        LSLB
7409 58        LSLB
740A EB 84      ADDB   ,X
740C AC 04      CMPX   ,U
740E 27 0C      BEQ    ETI4
7410 86 01      LDA    #01
7412 AA 42      ORA    2,U
7414 A7 42      STA    2,U
7416 1F 98      TFR    B,A
7418 30 01      LEAX   1,X
741A 20 E8      BRA    ETI
741C 33 42      ETI4   LEAU  2,U
741E 37 01      PULU   CC
7420 39        RTS

```

COMPARE AUX CARACTERES
AUTORISES

```

7421 36 20      COMPAR PSHU  Y
7423 108E 6C01  LDY    #VAL
7427 E1 A0      AR     CMPB  ,Y+
7429 27 08      BEQ    AV
742B 108C 6C17  CMPLY  #VAL+NVAL
742F 26 F6      BNE    AR
7431 1C FB      ANDCC  ##FB
7433 37 20      AV     PULU  Y
7435 39        RTS

```

CONVERSION HEX/ASCII

```

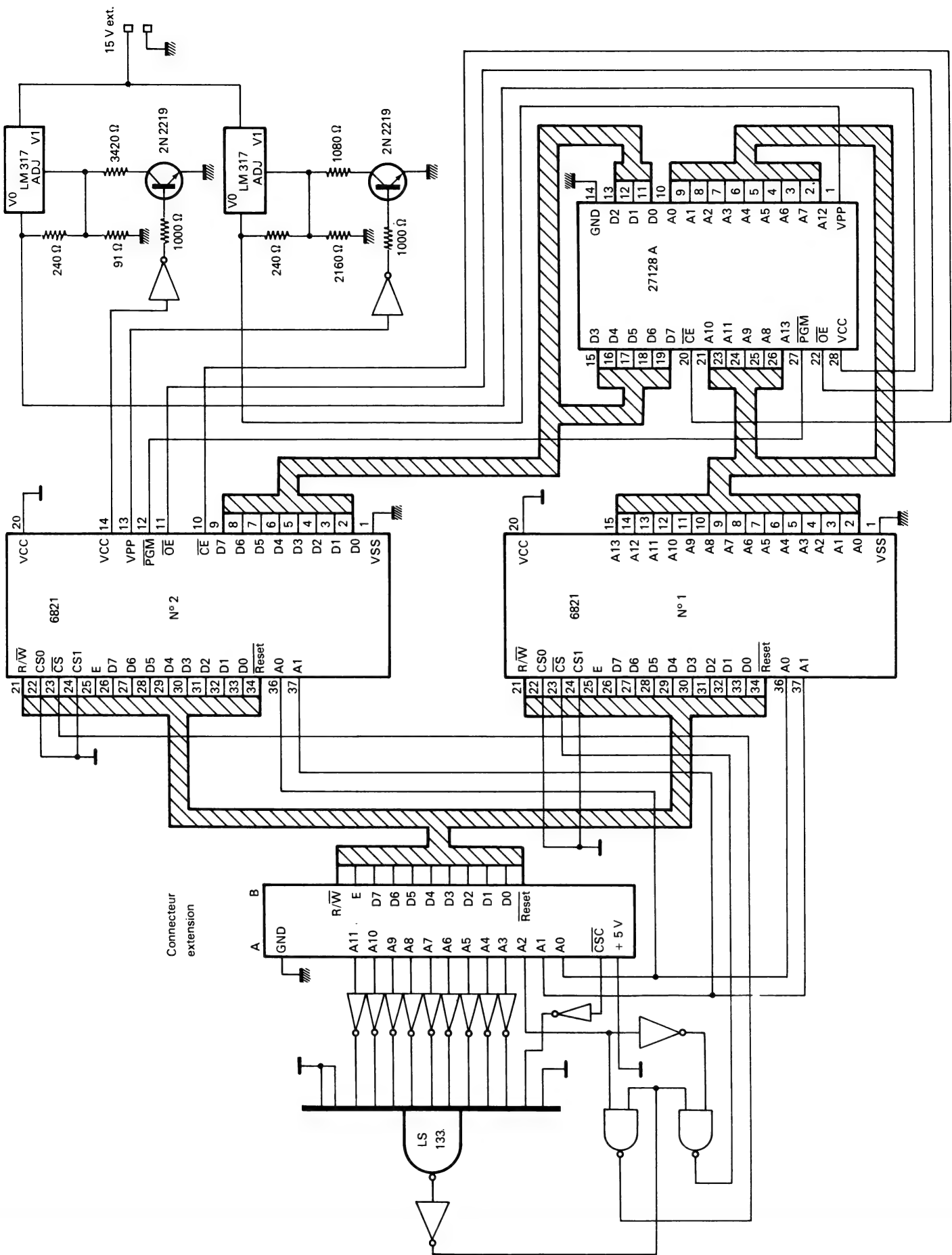
7436 36 16      HEAC   PSHU  D,X      Sauvegarde
7438 C6 FF      ENCO   LDB   #-1      Initialise B a 0
743A 5C        INCB
743B A6 05      LDA    B,U      Charge un octet
743D 8D 10      BSR    DEC
743F A6 05      LDA    B,U      ← Recharge le même octet

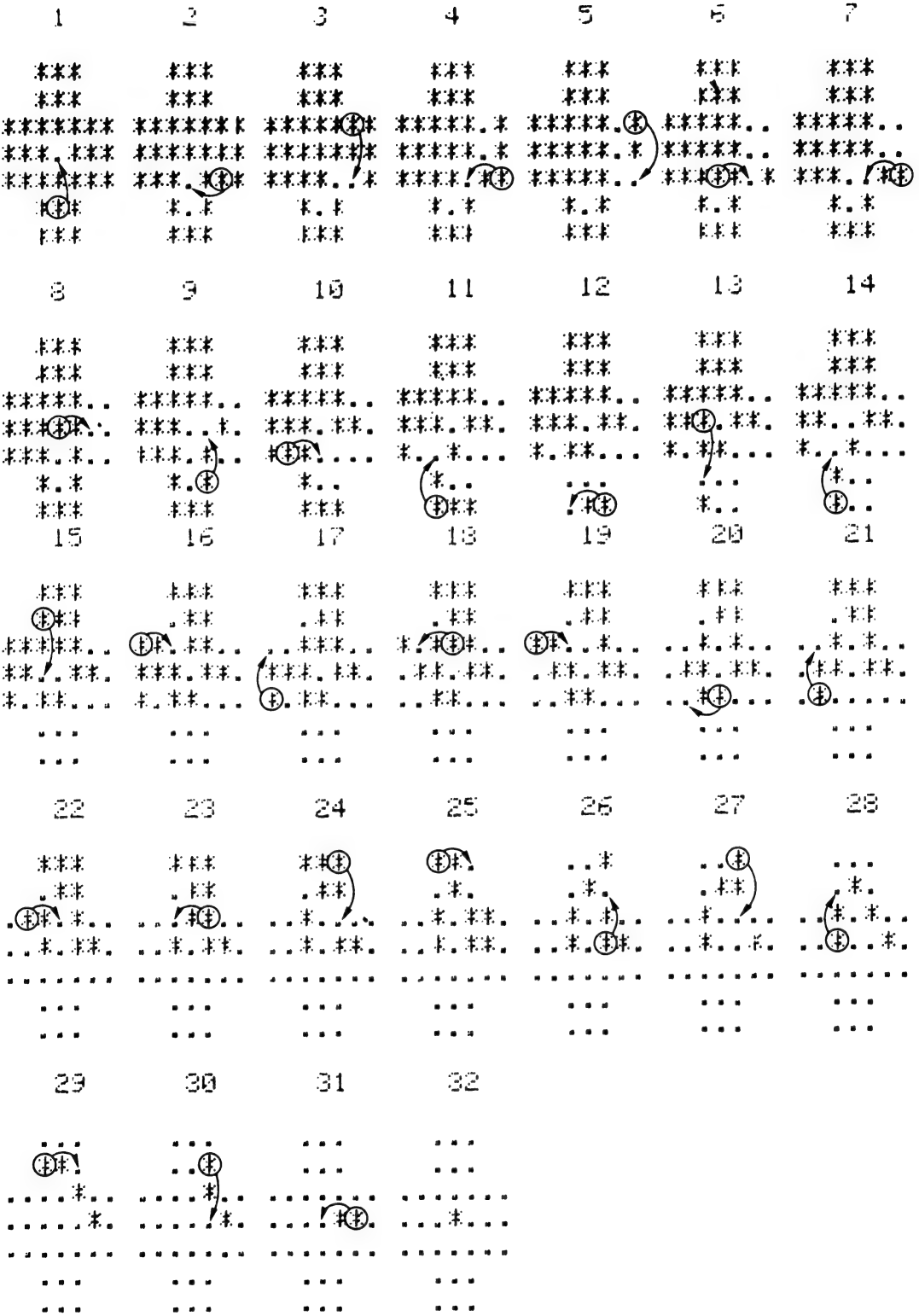
7441 84 0F      ANDA   ##0F      ← Masque le quartet de poids fort
7443 8D 0E      BSR    CHA1
7445 C1 01      CMPB   ##1
7447 26 F1      BNE    ENCO      ← Fin ? Non. On continue

7449 5F        CLRB      ← Oui. On charge dans l'octet final
744A E7 84      STB    ,X
744C 37 16      PULU   D,X
744E 39        RTS
744F 44      DEC    LSR

```

```
7450 44          LSRA
7451 44          LSRA
7452 44          LSRA
7453 88  30      CHAR1  ADDA  #$30
7455 81  39      CMPA   #$39      Plus Petit que 9 ?
7457 23  02      BLS    *+4      Oui. Alors fin
7459 88  07      ADDA   #$7      Non. on ajoute 7
745B A7  80      STA    ,X+
745D 39          RTS
          0000      END
```



Une solution du jeu du solitaire.

COMMUTATION DE BANQUES MÉMOIRES

* COMMUTATION DE BANQUES MEMOIRES *

* SELECTIONNE UNE DES 6 BANQUES *
* DISPONIBLES SUR T07/70 *
* LE REGISTRE A DOIT CONTENIR LE NO *
* DE LA BANQUE A SELECTIONNER *

6D00 86 01 DEBUT LDA #1 Banque no1
6D02 B0 6D06 JSR BANQ
6D05 3F SWI

6D06 34 6D06 EQU *
6D06 34 56 PSHS D,X,U
6D08 CE E7C0 LDU ##E7C0
6D08 E6 4B LDB 11,U
6D0D C4 FB ANDB ##FB

Force le bit 2 de E7CB à 0 (registre de contrôle du port B) afin d'accéder au registre de sens de transfert

6D0F E7 4B STB 11,U
6D11 8E 6D1E LDX #VAL

Pointe sur la table des valeurs de commutation

6D14 A6 86 LDA A,X

Charge l'octet pointé dans le registre de direction

6D16 A7 49 STA 9,U
6D18 CA 04 ORB ##04

Force le bit 2 de E7CB à 1

6D1A E7 4B STB 11,U
6D1C 35 D6 PULS D,X,U,PC

6D1E 0F 17 E7 67 VAL EQU *
6D22 A7 27 FCB \$0F,\$17,\$E7,\$67,\$A7,\$27
0000 END

Valeur de commutation

CASSETTE DE PROGRAMMES COMPLEMENTAIRES

Pour obtenir la cassette des solutions
aux exercices et utilitaires proposés ci-
dessous par les auteurs, veuillez rem-
plir le bon de commande page 239.

INVITATIONS A L'AVENTURE

Nous suggérons — sans sujétion — ci-après quelques sujets de programmes vous permettant de contrôler vos connaissances.

Nous ne donnons que les grandes lignes directrices.

Si vous désirez avoir la solution des auteurs, complétez le bon que vous trouverez à la fin de l'ouvrage pour recevoir la cassette. Elle contient, en outre, quelques « utilitaires » dont la description est donnée en fin de chapitre (vous pouvez évidemment essayer de les écrire).

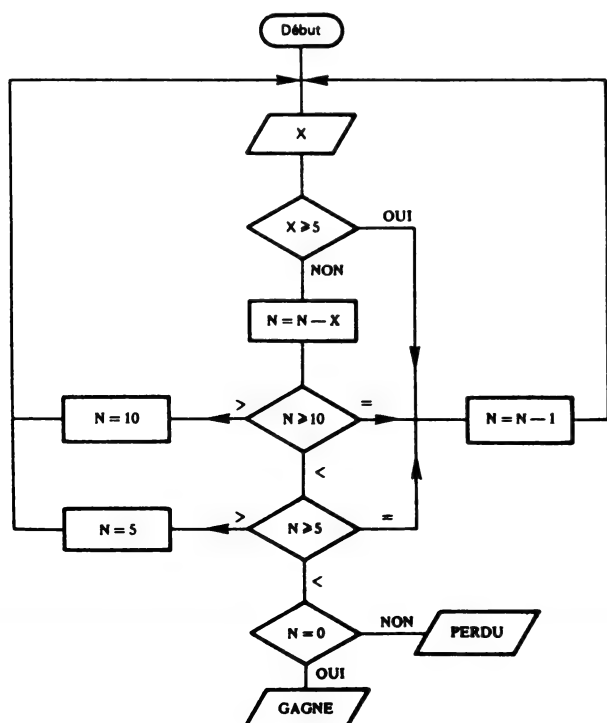
Ce jeu consiste à partir de 16 allumettes ; chaque joueur en retire au maximum 4 et celui qui retire la dernière a perdu.

Pour ce premier « test », nous vous donnons un organigramme et une méthode.

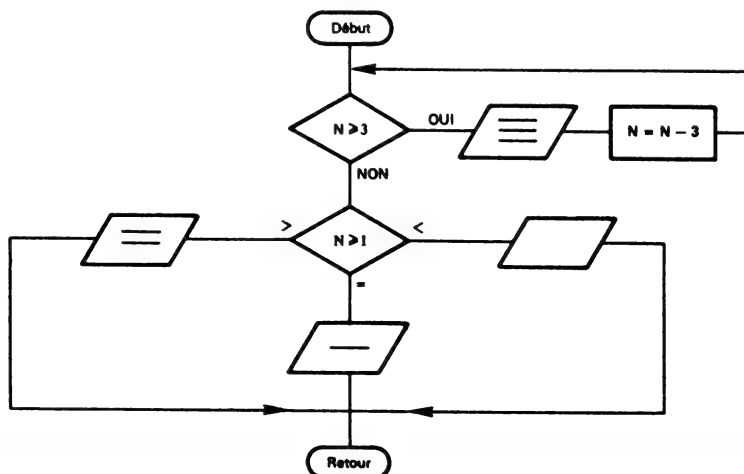
La méthode, *pour gagner*, consiste à *laisser à l'adversaire 6 allumettes*, d'où l'organigramme suivant pour la machine qui, très intelligente, opère un raccourci dès qu'il reste moins de 5 allumettes.

Au début, la machine joue si l'on presse une touche de valeur supérieure ou égale à 5 (X est la touche pressée) ; le nombre d'allumettes restant est $N + 1$.

1 JEU DE NIM



Organigramme du jeu de NIM.



Organigramme de l'affichage.

Pour ce programme, la difficulté réside dans l'affichage. Les allumettes peuvent être figurées par des segments horizontaux. Nous aurons donc au début 5 paquets de trois allumettes plus une :

```

— — — — —
— — — — —
— — — — —

```

qui disparaîtront petit à petit.

Nous vous conseillons de « temporiser » le jeu afin de bien suivre son déroulement.

2 JEU DE MARIENBAD

C'est un autre jeu d'allumettes qui compte également 16 objets, mais répartis en quatre paquets de 7, 5, 3, 1 allumettes. Dans cette variante du jeu célèbre de Marienbad, il faut retirer dans *un* paquet de 1 à 7 allumettes.

Pour tenter de gagner, il faut laisser à l'adversaire un nombre *impair* d'allumettes. Le perdant est celui qui retire la dernière allumette.

Problèmes :

- Affichage.
- Lecture du clavier : numéro du paquet, nombre d'allumettes à enlever.
- Détermination de la parité d'un nombre.

3 JEU DE MASTER-MIND

Il s'agit de trouver une combinaison de 4 chiffres compris entre 0 et 9 par essais successifs. La machine répond par le numéro de l'essai et le nombre de chiffres « bons et bien placés » (B/BP) et « bons mais mal placés » (B/MP). Le « choix » de la combinaison est effectué par tirage (voir programme du LOTO).

Ce tirage peut donner des doubles, par exemples :

1, 1, 8, 0

si on essaie « 1, 2, 8, 0 », la machine peut répondre (hypothèse de travail) :

XE 30 : Xème essai 3 : B/BP, 0 : B/MP

Par contre, à l'essai « 2, 9, 1, 4 », la machine répondra :

YE 02 : Yème essai 0 : B/BP, 2 : B/MP

L'essai est bon si la machine répond : UE 40

Conseils

- Commencer par les B/BP.
- Prendre des précautions en cas de doublets.
- Chercher les B/MP.

4 FUSEE

Vous êtes à bord d'un vaisseau spatial que vous désirez poser sur un astre inconnu. Sur l'écran, vous affichez l'altitude (de 0 à 9999 unités) et la vitesse initiale de descente (de 0 à 99 unités). C'est parti ! Régulièrement la distance diminue d'une quantité égale à la vitesse (une temporisation fixe la périodicité de l'opération).

Vous pouvez diminuer la vitesse en appuyant sur une touche numérique du clavier, fixant une décélération constante (diminution régulière de la vitesse) à moins que vous n'appuyiez sur la touche 0.

Il faut arriver à vitesse nulle (00) à une altitude nulle (0000), sinon :

- on reste en l'air ! (xxx 00)
- on s'écrase !! (0000 xx).

Problèmes

- Soustraction décimale.
- Affichage.
- Périodicité.
- Messages de louanges ou d'injures suivant le résultat.

5**TIR**

(avec photostyle)

Un pavé apparaît de façon aléatoire sur l'écran, vous avez quelques instants pour le toucher avec le photostyle. Après apparition de 10 pavés, le score s'affiche.

Problème

- Générer X et Y (position du pavé) : voir tirage du LOTO.
- Temporiser.
- Viser au photostyle : voir SOLITAIRE.

Remarque : on peut émettre un BIP pour éveiller l'attention du tireur.

6**PIANO**

Le clavier de votre ordinateur compte 43 touches codées ASCII dont vous pouvez doubler le nombre à l'aide de la touche

Il est conseillé de se mettre au préalable en « minuscules ».

Il s'agit d'attribuer à chaque touche une note, vous disposez donc de 86 notes, soit plus de 7 octaves !

Les notes seront toutes de même durée.

Vous aurez à écrire 2 tables, l'une donnant la « note », l'autre l'octave. A moins que vous ne préfériez un traitement mathématique.

7**UTILITAIRES
BASIC**

Pour tous les programmes en Assembleur proposés jusqu'à maintenant, nous avons supposé que vous disposiez de la cartouche ROM « ASSEMBLEUR ». Mais certains lecteurs peuvent être tentés par l'écriture directement en mémoire (code hexadécimal ou langage machine) et sous BASIC, des programmes proposés.

D'autres peuvent être tentés d'écrire en binaire des données ou de petits modules de programmes — ce qui se fait souvent — qu'ils appelleront sous BASIC à l'aide des instructions :

EXEC <adresse-mémoire>

ou USR <numéro> <argument> précédé de

DEF USR <numéro> <adresse-mémoire>

et cela sans bourse délier...

L'utilitaire BASIC fait appel à un MENU qui autorise 5 tâches essentielles :

a) *LIST*. Permet d'afficher à l'écran, en hexadécimal, le contenu des cases mémoires comprises entre deux bornes qui sont :

Adresse de départ et Adresse de fin

b) *MODIFICATION*. Permet de visualiser et de modifier le contenu d'une case mémoire. La validation de la modification par la touche ENTREE visualise la case suivante.

c) *ECRITURE*. Ce choix est à rapprocher du choix précédent excepté qu'il ne renvoie pas le contenu de la case mémoire désignée.

d) *EXECUTION*. Ce choix permet de faire exécuter un programme objet (en code machine) préalablement chargé en mémoire.

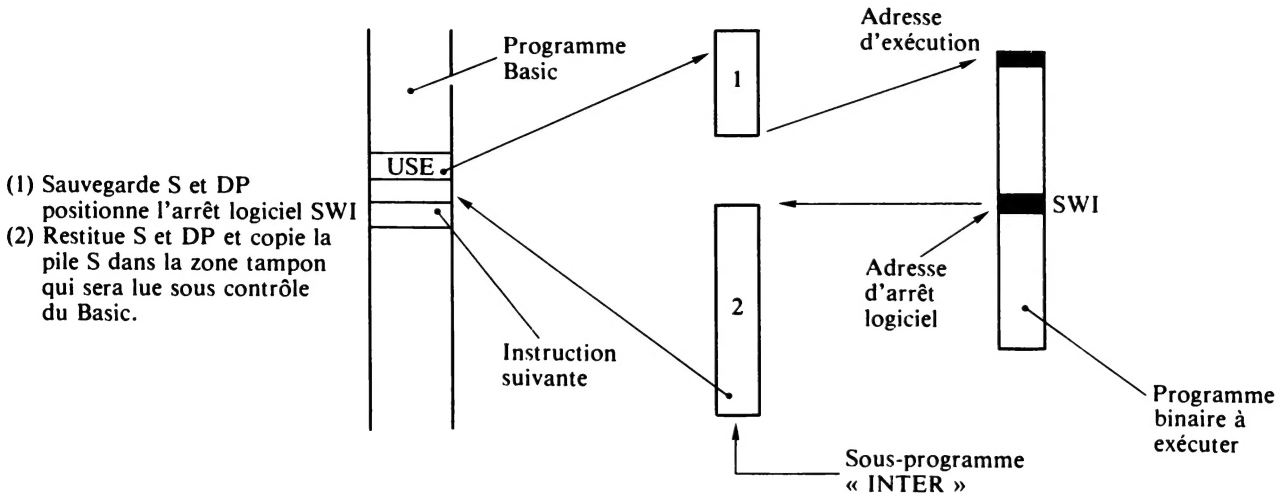
e) *BASIC*. Permet le retour sous le contrôle de l'éditeur BASIC.

Problèmes

Il n'y a pas de problèmes majeurs pour les tâches a, b et c qui peuvent se résumer à un « ballet » de PEEK et POKE bien placés.

La tâche d (*EXECUTION*) est plus délicate car il faut exécuter un programme sans en modifier l'écriture et en respectant les impératifs du BASIC qui sont : sauvegarde du pointeur S et du registre DP.

On peut imaginer la structure suivante :



Autre difficulté :

Où stocker le programme que l'on souhaite faire exécuter sous contrôle du programme BASIC ? Si vous utilisez une disquette, le DOS accepte 8 K octets et le programme BASIC est toujours rangé **après** le DOS.

Le programme utilitaire BASIC se termine vers l'adresse 9500 et utilise le sous-programme INTER implanté en fin de RAM à partir de BFA9. Vous disposez donc, pour écrire votre programme, de la zone mémoire comprise entre 9500 et BFA9.

8

LECTURE/ ECRITURE

de valeurs binaires
depuis/vers
un périphérique

On peut être amené à lire ou écrire, en cours d'exécution d'un programme, des valeurs binaires stockées sur une disquette ou une cassette. Il s'agit des instructions BASIC : SAVEM et LOADM.

Comment faire en Assembleur ?

Réponse

Créer les sous-programmes équivalents en Assembleur.

Problème

Reconnaître à la lecture et respecter à l'écriture le format MICROSOFT afin que ces fichiers puissent être relus par les routines habituelles sous contrôle du MONITEUR ou de l'EDITEUR. Les programmes de la disquette ou de la cassette portent les noms suivants :

LOADK7	: Lecture de la cassette
SAVEK7	: Ecriture sur la cassette
LOADIS	: Lecture de la disquette
SAVDIS	: Ecriture sur la disquette

Nota : Le registre X pointe le nom avec suffixe du fichier à lire ou écrire. Les six (6) octets suivants représentent les adresses de début, de fin et d'exécution. Soit :

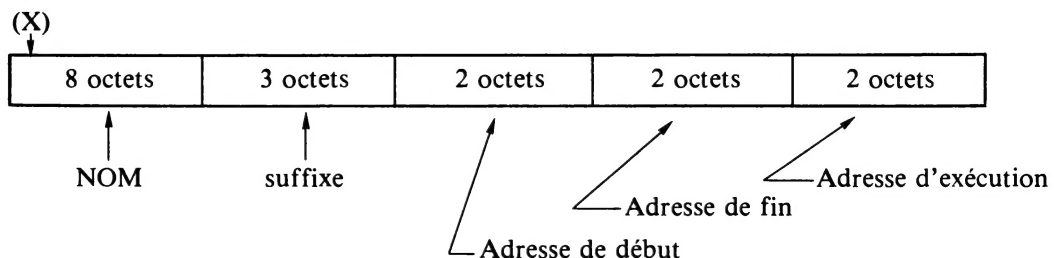


TABLE DES MATIÈRES

1 — Addition hexadécimale 1 digit	12
2 — Addition hexadécimale 4 digits	19
3 — Soustraction hexadécimale 4 digits	26
4 — Multiplication décimale de 2 nombres de 1 chiffre	33
5 — Conversion DCD-hexadécimal, nombre de 2 chiffres	41
6 — Multiplication décimale de 2 nombres de 2 chiffres	44
7 — Multiplication décimale de 2 nombres de 2 chiffres, adressage direct ..	50
8 — Entrée/affichage, nombre de 2 chiffres	52
9 — Message	55
10 — Affichage d'un nombre de 4 chiffres	57
11 — Multiplication décimale de deux chiffres - entrée clavier - affichage ...	59
12 — Multiplication hexadécimale signée	62
13 — Somme décimale de N nombres de 2 octets	66
14 — Chargement d'un message	69
15 — Chargement d'une table de nombres de 16 bits	70
16 — Soustraction décimale nombres de 2 chiffres	74
17 — Conversion DCB-Hex, nombres de 16 bits	76
18 — Conversion Hex-DCB, nombres de 16 bits	80
19 — Conversion DCB-Hex, pour nombres non entiers	83
20 — Conversion Hex-DCB, pour nombre non entiers	87
21 — Multiplication hexadécimale avec virgule	90
22 — Division hexadécimale 16 bits par 8 bits quotient non entier	93
23 — Puissance décimale	97
24 — Parité	100
25 — Affichage avec table de codes	103
26 — Choix de programme au clavier	105
27 — Tri	108
28 — Dictionnaire	111
29 — Affichage semi graphique	119
30 — Graphisme	121
31 — Graphisme mathématique sinusöide	124
32 — Graphisme mathématique cercle	128
33 — Jeu du solitaire	132
34 — Un nouvel affichage	138
35 — Rectangle vide en mode graphique	141
36 — Rectangle vide en mode caractère	143
37 — Rectangle plein en mode graphique	145
38 — Rectangle plein en mode caractère	148
39 — Pointeur visible	152
40 — Caractères TELETEL	157
41 — Séquence US	163
42 — Séquence ESC	167
43 — Création d'un nombre aléatoire	172
44 — Tirage du Loto	176
45 — Génération d'une note	180
46 — Boîte à musique	183
47 — Carillon de porte	188
48 — Journal lumineux	196
49 — Bataille navale	201
50 — Programmation de mémoire EPROM	210
51 — Commutation de banques mémoires	233
Programmes complémentaires	234

POUR ALLER PLUS LOIN :

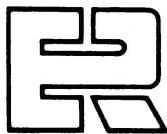
DES PROGRAMMES COMPLEMENTAIRES SUR CASSETTE POUR VOTRE T07/70



contenu détaillé
voir page 233

1. Jeu de NIM
2. Jeu de Marienbad
3. Jeu de Master-Mind
4. Fusée
5. Tir (avec Photostyle)
6. Piano
7. Utilitaires
8. Lecture/Ecriture de valeurs
binaire depuis/vers périphériques

S. E. C. F.



ÉDITIONS RADIO

9, RUE JACOB - 75006 PARIS
Tél. (1) 329.63.70

Demandez cette cassette chez votre revendeur habituel ou profitez du bon de commande ci-dessous.

BON DE COMMANDE

A adresser à : **S.E.C.F. Editions Radio, 9, rue Jacob 75006 Paris**

Je désire recevoir par la poste au prix de 170 F, la cassette de programmes complémentaires sur T07/70 (Port et emballage compris.).

Nom : _____

Adresse : _____

Prénom : _____

Ci-joint règlement à l'ordre de : **S.E.C.F. Editions Radio**

Chèque postal 3 volets
sans indication de N° de compte ☐

Chèque bancaire ☐

Mandat postal ☐